Adrian Kosmaczewski

Full website archive

1996-2023

JJJ

Adrian Kosmaczewski

1996-04-06

L'histoire humaine est pleine de surprises. Prenons, par exemple, le cas tres particulier des "Multiplication des J". En effet, de très nombreuses personnalités connues par leurs réalisations dans de très nombreux domaines possèdent des patronymes commençant par la lettre J, considérée d'ailleurs par certaines castes de chamans du sud de la polynésie comme miraculeuse.

Citons, à titre d'exemple, les noms suivants:

- **John Jackson**, neurologue britannique (1834-1911), auteur d'importants travaux sur l'épilepsie;
- **Jonathan Jackson**, général américain (1824-1863), connu par son surnom de Stonewall, et par son activité en tant que sudiste dans la guerre de Sécésion, mais tué accidentalement par ses troupes;
- **Jacob**, patriarche de l'Ancien Testament (très longtemps-très longtemps plus le temps qu'il ait vécu, le tout en négatif); il y a eu aussi **Joël**, **Job**, **Jésus**, **Joas**, **Jonas**, **Joseph**, **Jean**, **Judas**, et tant d'autres...
- **Jacob Jacobi**, mathématicien allemand (1804-1851), connu universellement par sa démonstration sur la double périodicité des fonctions elliptiques, dont il a montré la possibilité de les résoudre via les archifameuses fonctions theta, dont tout le monde (scientifique) se doutait de la réponse et se demanda pourquoi n'y avait-on pas pensé auparavant;
- **Jacob I**, **Jacob II**, ..., **Jacob VII**, rois d'Angleterre, Ecosse, Irlande et tout ça;
- **Jens Jacobsen**, nouvelliste et poète danois (1847-1885), assez connu (si quelqu'un a entendu parler de ce bonhomme, faites savoir);
- **Jaume Jacomart**, peintre valencien (1413-1461), dont une seule oeuvre a été trouvée à ce jour;
- **Jean-Joseph Jacotot**, pédagogue français (1770-1840);
- **Joseph Jacquard**, mécanicien français (1752-1834);
- **Jules Janssen**, astronome et physicien français (1824-1907), fondateur de l'observatoire de Meudon, découvreur de l'hélium dans le spectre solaire, et inventeur du compas aéronautique;
- **Juan de Jáuregui**, peintre espagnol (1583-1641), fameux par sa réfutation doctrinale du gongorisme et sa manifestation

théorique du conceptisme, uniques en son genre;
- **Jean Jaurés**, politicien français (1858-1914), socialiste anti-marxiste, fondateur de “L’Humanité”, assasiné par son hostilité à la guerre;
- **Jeep**, véhicule de la firme Willis (1939-1988), fameux par ses amortisseurs, récemment détroné par son successeur “Zoomer”;
- **Johannes Jensen**, écrivain danois (1873-1950), prix Nobel de littérature en 1944;
- **Jerome Jerome**, écrivain anglais (1859-1927), auteur humoristique (le nom contribuait largement au succès de ses livres, je suppose);
- **Jacinto Jijón**, arqueologue et politicien équatorien (1891-1950);
- **Jerónimo Jiménez**, compositeur espagnol (1858-1923);
- **Juán Jiménez**, poète espagnol (1881-1958);
- **Jordi Joan**, sculpteur catalan (?-1418);
- **Jonathan Joachim Jodry**, un copain de l’uni (1973-?);
- **Joseph Joffre**, militaire français (1852-1931), maréchal de france;
- **James Jones**, écrivain américain (1921-1977);
- **Jacob Jongelink**, sculpteur flamand (1530-1606);
- **Johan Jongkind**, peintre hollandais (1819-1891);
- **Jacob Jordaens**, peintre flamand (1593-1678);
- **Jonas Jørgensen**, chimiste danois (?-?), professeur à l’Université de Genève;
- **Joseph Joubert**, écrivain et moraliste français (1754-1824);
- **James Joule**, industriel et physicien britannique (1818-1889);
- **Jean Jouvenet**, peintre français (1644-1717);
- **James Joyce**, écrivain irlandais (1882-1941);
- **Juan de Juni**, sculpteur espagnol d’origine française (1507?-1577);
- **Jupiter**, dieu romain, (toujours-jamais), connu par sa planète natale;
- Et finalement **James Jurin**, médecin et physicien anglais (1684-1750).

Une Balade Sur Les Meilleurs Sites

Adrian Kosmaczewski

1996-11-22

Pour les étudiants universitaires de Genève: clickez ici[1] si vous avez un accompte sur SC2A.UNIGE.CH ou UNI2A.UNIGE.CH pour connaître mieux votre accès à Internet.

Une balade sur les meilleurs sites

Article paru au “V Magazine”, le magazine du Nouveau Quotidien, le 22 novembre 1996, après le dossier “Internet: les questions que vous n’osez plus poser”, par Gabriel Singrist[2].

Grâce à son évolution fulgurante, le Web est devenu un formidable espace de découvertes, d’évasion et de distraction, Comme avant-goût, nous avons sélectionné une brochette d’adresses répertoriées en rubriques.

Les sites “point de départ”

Pour commencer un bon surf, une bonne idée est de se diriger vers des sites regroupant déjà des adresses par thèmes. En Suisse, la plupart des fournisseurs d’accès comme SwissOnLine ou VTX offrent leurs sélections. Un excellent exemple de point de départ est le Guide Internet qui contient une liste impressionnante de liens. Au Québec, Planète propose aussi toute une palette d’adresses francophones. Les Français ne sont pas en reste avec le très complet Club Internet ou Infonie. Une liste des serveurs suisses existe aussi.

Les moteurs de recherche

Un autre bon moyen pour bien commencer son voyage est d’utiliser les moteurs de recherche. Ce sont de puissants outils pour retrouver de l’information en fonction de mots ou de phrases-clés. On les divise en deux catégories: les “araignées”, utilisées pour trouver un lien sur un document précis, et les annuaires qui regroupent les sites

[1] /blog/manuel-pratique-de-lutilisateur-dinternet-et-du-vms-à-sc2a.unige.ch-et-à-uni2a.unige.ch/

[2] mailto:gabriel.singrist@edipresse.ch

par rubriques. Par exemple, pour retrouver le livre du Docteur Gubler sur François Mitterrand, on utilisera un moteur de type “araignée”. Par contre, pour retrouver l’ensemble des sites qui parlent de jazz, d’histoire ancienne ou de tourisme, on utilisera plutôt un annuaire. Parmi les araignées, on peut citer Altavista, le plus connu, le nouveau moteur rapide d’InfoSeek, et, en vrac, Excite, HotBot, et, en Suisse, SwissSearch. Parmi les annuaires, le plus complet est sans conteste le célèbre Yahoo! Pour les adresses suisses, l’annuaire s’appelle SwissCom. Certains moteurs peuvent effectuer les deux types de recherches, comme Lycos et Search. Certains proposent des recherches ciblées, par exemple francophones, comme sur Nomade, ou parmi les sites sélectionnés pour leur qualité, comme le célèbre Magellan, une des références du Net. Une liste des moteurs de recherche répertoriées par pays se trouve sur EDirectory. De manière générale, il faut être vigilant à la manière de formuler une recherche. Par exemple, il ne faut mettre de majuscules que pour les noms propres et entourer les phrases de guillemets. On peut souvent cumuler des arguments en les séparant d’un espace ou d’un signe +. Un exemple de recherche avec une araignée serait d’introduire ” le grand secret”+ Gubler. Pour restreindre l’espace de recherche à un pays, on peut souvent rajouter les deux lettres qui le symbolisent au début de la requête, comme ceci: host:ch”le nouveau quotidien”

Bandes dessinées

Les revues comme Fluide Glacial, Yoko, ou Mang@Cult sont d’excellentes sources pour les bédéphiles. A part les éditeurs comme Dargaud, il existe aussi un musée et un paradis de la BD, lieu de rencontre réalisé par les dessinateurs. Les 100 ans de la BD se fêtent sur Imaginet.

Business

Beaucoup de services consacrés aux affaires sont payants ;-). Les journaux économiques ont tous leur site: La Tribune Desfossés, Les Echos, le Financial Time, ou le Wall Street Journal. Sinon, on trouve par exemple le célèbre DowJones, Reuters, et Business Strategies. CNN possède une rubrique financière complète et fréquentée.

Cinéma

Les programmes ciné de Suisse Romande sont sur Edicom. On peut suivre le festival de Venise, consulter des critiques internationales ou des bases de données d’acteurs et de films.

Culture

Il existe un foule de musées sur le Net. Certains sont devenus des musts, comme le réseau WebMuseum, ou le Metropolitan Museum

de New York. Le site officiel de l'expo des portraits de Picasso au Grand Palais à Paris comporte photos et biographie. On trouve aussi des galeries new-yorkaises, un site d'architecture ou le musée virtuel d'Andy Warhol.

Dictionnaires, Références

Angoissé devant un texte intraduisible ? Logos arrive à la rescousse pour traduire à peu près n'importe quelle langue. Eurodico est un prototype de traducteur qui fonctionne aussi pas mal. Dans presque chaque pays (sauf la Suisse qui attend les PTT), on trouve aussi les annuaires téléphoniques, comme sur InfoSpace pour les Etats-Unis et le Canada. Le succès de l'horaire des CFF s'explique par sa rapidité et son efficacité.

Evasion

Pour changer d'air, rien de tel qu'une balade en famille à travers le zoo de San Francisco ou dans le Système Solaire. Un voyage dans la capitale française, en Italie, ou dans une grande ville, peut s'organiser sur le Net. Fatigué d'une longue journée de travail ? Expérimentez avec le tourisme virtuel ou suivez la mission de la Nasa sur la planète Mars. Marre de la grisaille ? Départ pour Tahiti ou Hawaï. Avant de skier cet hiver, n'oubliez pas de vérifier les conditions d'enneigement en Suisse et la météo! Pour les voyages, Swissair propose horaires et offres spéciales. Les agences comme SSR ou Artou proposent aussi des vols de dernière minute à prix cassés.

Job

Trouver un boulot sur Internet, c'est le dernier truc à la mode. En Suisse, on peut dénicher une place d'assistant ou de chercheur sur Telejob. Une bonne idée est d'y laisser une annonce avec un CV. Pour trouver du boulot à l'étranger, on peut essayer le Réseau Européen pour l'emploi, JobNet ou JobCenter. Beaucoup de boîtes mettent aussi des annonces directement sur leur site.

Messageries

Le Web, c'est aussi les dialogues et les rencontres. Un petit tour dans l'ambiance feutrée de The Palace, une messagerie multimédia de tout premier ordre, que nécessite cependant un logiciel spécifique à télécharger sur place. Pour entrer dans le monde des messageries IRC, il faut aussi s'équiper d'un logiciel adéquat. Une liste des messageries autour du monde est disponible.

Mode

Les serveurs de référence, avec défile on-line, sont Fashion Internet aux Etats-Unis et Fashionlive en France. FirstView contient une vaste collection de photos de mode. La plupart des maisons dans le vent ont leur site: Levi's Strauss, Benetton, W&LT, Paul Smith ou APC. Un magazine de mode électronique, @Mode, existe à côté de ténors comme Elle. Une liste des meilleurs sites consacrés à la mode se trouve sur FashionAngel.

Musique

Des nouveautés du disque classique à la scène musicale alternative suisse en passant par les grandes maisons de disques et les magazines comme Les Inrockuptibles, les sites musicaux pullulent. Le très complet Jazz Web pour le jazz et l'original Firefly sont parmis les incontournables. D'autre part, les stars on presque toutes leur site: Björk, Suzanne Vega, les Young Gods, Oasis, etc. Prince est probablement un des plus actifs sur le Web.

Sciences et Santé

Les grands magazines scientifiques comme Science, Nature ou Scientific American ont tous leur site. On trouve aussi des encyclopédies de médicaments, comme le Vidal français, des sites consacrés à la prévention, comme Carrefour prévention ou ActUp. L'Institut Suisse pour la Santé Publique renseigne par exemple sur la grippe. L'hebdomadaire des pharmaciens suisses Optima propose aussi de nombreuses rubriques santé.

Technologies

Le Web possède un langage informatique, baptisé Java qui permet d'animer des pages avec des applets, répertoriées sur Gamelan. Le magazine Javaworld parle des dernières trouvailles. Une des nouveautés à la mode est la radio sur Internet, avec RealAudio. Shockwave permet de fabriquer des animations.

Voitures

Les grandes marques présentent leurs nouveaux modèles sur le Web: BMW, Mercedes Benz, Opel, Toyota, Ferrari, Fiat, Volvo, etc. On trouve aussi des guides de référence et des voitures de rêve.

Divers

Quelques bonnes adresse pour l'oenologue averti: The Wine Vineyard, La Place du Vin et The World Wine Web. Pour les petits plats qui vont

avec, faites un saut chez Epicuria et goûtez son excellente tarte tatin ! A mentionner en vrac: Msg. Gaillot, les Guignols de l'Info, les jeux, et les logiciels gratuits. Pour les petits, "Mes premiers pas sur Internet" est un excellent site français destiné aux enfants, aux instituteurs et aux parents.

Erotique

L'attente faisant partie intégrante de l'érotisme, c'est bien connu, les meilleures adresse érotiques sont celles que l'on trouve soi-même ! Quoi de plus excitant en effet que de lancer une recherche croustillante… et de voir apparaître lentement les résultats, comme dans une sorte de strip-tease virtuel. Pas besoin de vous aider pour les mots-clés. ;-)

Manuel pratique de l'utilisateur d'Internet et du VMS à SC2A.UNIGE.CH et à UNI2A.UNIGE.CH

Adrian Kosmaczewski

1997-03-03

VERSION 5.4, du 3 mars 1997.

Figure 1: Calvin & Hobbes, (c) par Bill Waterson

Pour une promenade en Internet un peu plus commentée, clickez ici[1].

Message RIDMI.TXT:

Et voila, encore une mise a jour. J'espère qu'elle sera utile a tous. Cette fois, la distribution de ce Manuel ce fait via le WEB, ce qui veut dire que bien plus que 50 millions de malheureux peuvent lire ce projet d'oeuvre d'art. Mais la plus grande nouveauté de la rentrée est, sans aucun doute, la possibilité de faire son propre URL sans demander de l'aide à personne - oui, bon, il vous faudrait au moins un éditeur de HTML, mais avec WORD tout s'arrange. Et pis, regardez comme c'est bien quand c'est bien fait Florian a une page HTML exemplaire, je dirais même unique. Celle de Andre Stefanov n'est pas mal non plus. Et que dire alors de celle de notre cher GloGlo ! Sans parler de celle, lamentablement en cours de développement, de Hendrik Stattman! Des larmes, j'te jure. En plus de cela, la trouvaille géniale des fichiers

[1] /blog/une-balade-sur-les-meilleurs-sites/

batch, et du fichier batch LOGIN.COM qui s'exécute automatiquement chaque fois que vous loggez dans le système ! Ce LOGIN.COM est indispensable pour que les "Signature Files" marchent comme il le faut ! A lire absolument !

Prologue

Ceci est une nouvelle mise a jour du légendaire fichier "INTERNET.TXT" que plusieurs d'entre vous avez reçu entre 1994 et 1995 via E-MAIL et dont les mérites vous avez vantes de gauche a droite. Voici donc une version légèrement modifie, revue, augmentée, corrigée, adaptée au WWW, de ce manuel tellement pratique a avoir sous l'écran. J'espère que son succès ira en grandissant, mais pour cela vous êtes pries de bien vouloir verser la modique somme (moins modique qu'avant) de 300 Sfr sur mon account (faites PUT 300 SFr dans le serveur anonymous adrian.veut.lefric.dabord.ondiscute.ensuite), ceci en guise de remerciement "à la shareware", et pour m'inciter à continuer sur cette voie de sagesse et lumière qu'est l'Internet. Merci d'avance. Muchas gracias, si senior, muy bien. J'adore parler l'espagnol comme les suisses, c'est tellement marrant. Il est a souligner que ce manuel n'est valable que pour SC2A.UNIGE.CH et UNI2A.UNIGE.CH, pas qu'on me sorte la combine comme quoi y'a qu'des conneries sur ce fichier, eh ben moi j'lui dis pourquoi tu lis pas le titre, tu voas bien que c'est pas pour n'importe qui non ou bien, non mais pour qui y'se prennent, dedieu. Ah, j'ai une question, est-ce qu'il faut que je poste ceci a ug.comp ou a fr.rec.humour ???

Note Technique:

Dans ce qui suit, "nom" et "mot de passe" se référeront a, par exemple:

Nom

Mot de passe

fannen

starwars

glauser

volare

chanson4

jovanotti

barras3

visa-etc

musolino

vaniac

unal

turkyl

vaistij

alguita

ivancev2

marko

stefanov

theatre

On voit que la liste des abonnes grandit jour après jour. Tant pis pour eux. Pour ce qui seraient intéressés, je ne possède plus de version Word de ce document, celle que vous pouviez obtenir gracieusement sur FTP anonyme adrian.kosmaczewski.physique. Lamentablement peu de monde a profité de cette possibilité tellement inutile. Si ca continue, je vais abandonner les études pour mettre en place des serveurs FTP pour vendre des slips virtuels. La, maintenant, vous n'avez qu'à prendre une copie de ce fichier via Netscape... :^D J'adore ce smiley. Et en parlant de smileys, regardez plus bas, y'en a plein par ici.

Espace publicitaire:

Vous aimez ce manuel, hein?

Pour se connecter:

Depuis les VAX:

Allumez la machine, tapez T SC2A (ou T UNI2A) et ensuite vos nom et mot de passe. T pour TELNET, donc. Certains terminaux VAX ont un menu d'entrée préparé. Autant dire que la procédure de loggage, loggation ou de logginanisation s'en trouve amplement facilitée.

Depuis les PC's de Sciences I:

Ah, voilà, tout s'écorce un jour. Il faut se munir d'une photo passeport, d'un peu de patience, et foncer vers le troisième étage d'Uni Dufour, au bureau 307, oú M. Vuilleumier (téléphone interne 7636) vous donnera, après avoir signé un formulaire ad hoc, un login vous donnant accès aux PC se trouvant dans les différentes salles de Sciences I.

Revenu au pavillion des Sciences, l'heureux propiétaire du login allumera une machine, de préférence un Pentium (au fond du premier étage de SCI), et y introduira son login personnel. Il est à noter que pour cause de Duke Nukem's intempestifs, les logins génériques Intro, Biol, Chimie... ont lamentablement disparu pour toujours jamais. Arrivés à Windows, double-cliquez sur le groupe "Applications TCP-IP" et sur la première icône tout en haut à gauche de cette fenêtre (je crois que c'est un truc du genre TNVT, non ?). Ensuite sélectionnez SC2A.UNIGE.CH sur la liste déroulante, et loggez avec votre nom et mot de passe (celui pour SC2A, qui peut ne pas être le même que pour les PC's... !!!). Attention: ces logins là (pour SC2A) sont à demander à votre responsable de section, pas à M. Vuilleumier ! Mais bon, si vous êtes arrivés à lire ce fichier, pas de doute, vous avez sûrement soit l'un soit l'autre login. Bravo.

Depuis les MACs, les CRAYs ou les SUNs:

Hein, rigolo, y'en a pas... Oui, enfin, dans les départements des différentes sections y'a quelques Mac's qui traînent d'ici là, mais enfin bof, l'accès est bien sûr TRES restreint ! Il paraît qu' UNI MAIL y'en a encore plus. Tant pis pour eux.

A quoi cela nous mène-t-il ? (petite introduction au VMS):

Vous arriverez ainsi au chéri et amiable dollar $. Tapez HELP DOLLAR pour savoir ce que c'est. Vous pouvez y faire notamment du DEL ou DIR mais n'essayez ni COPY ni CREATE /DIRECTORY: trop compliqué. Vous pouvez par contre faire FINGER et voir qui se trouve de l'autre cote de l'écran... Mais attention, que si jamais quelqu'un vous PHONE, vous avez deux possibilités: soit vous faites PHONE et ensuite ANSWER, soit vous faites PHONE et puis REJECT...! Mais il se peut aussi que quelqu'un veuille TALKer avec vous, et la, ca devient grave... Mais ne vous affolez pas, vous pouvez toujours EDITer un fichier par ASCII, un fichier par la... Et si vous ne savez pas comment revenir au $, faites CTRL+Z ou CTRL+C autant de fois qu'il faudra, au pire, vous arriverez au DOS (oui, au pire).

Pour faire du World Wide Web en mode graphique sur Windows ou Mac:

Double-cliquez sur l'icône de NETSCAPE. Suivez ensuite les instructions affichées a l'écran (y'en a !). Pour les nombreux argentins du coin, sachez que La red Argentina est le serveur officiel des champions du monde de l'inflation. Parait-il, il est même possible de lire, avec NETSCAPE, les NEWS sans avoir a entrer dans son account. Mais vous voulez un conseil ? Utilisez NETSCAPE, MOSAIQUE a cote c'est du jus de manifestation d'escargots écrabouillée par un

troupeau d'éléphants équipés de marteaux pneumatiques. (violent, hein ? Pour 300 francs vous auriez la suite...) Par contre certains JODRY ont déjà fait l'expérience douloureuse de devoir fermer leur session NETSCAPE chez Playboy car leur copine pointe le bout du nez dans le bocal. Merci QUYNH !!! De toute façon, ils y ont coupé l'accès, et on ne peut plus acceder à d'autres servers que ceux de l'Uni... même pas à Yahoo, pour te dire.

Pour faire du WWW en mode texte sur VMS:

Rien de plus facile, tapez LYNX a cote du dollar et promenez vous avec les touches du curseur. En effet, c'est bien plus simple qu'avec la souris, vous n'avez pas a quitter les doigts du clavier. Et ca plante moins souvent qu'avec Windoze. André arrête d'emmerder, nomdedieu! N'oubliez jamais de revenir regulièrement sur l'URL de KosMik, qui a toujours une surprise!

Pour faire sa page WWW personnelle:

Tout d'abord, lire ICI, sur le serveur WEB de SC2A, les instructions détaillées. Ensuite, au boulot !

En résumé:

A:> FTP SC2A

FTP> bin FTP> mput *.* FTP> exit

C:> C SC2A

login: piratago password: ******

$ create /directory [.public_html] $ copy *.htm [.public_html] $ copy *.wav [.public_html] $ copy *.gif [.public_html] $ copy *.jpg [.public_html] **$ set file/prot=w:r public_html.dir , [.public_html]*.*** $ del *.gif;* $ del *.wav;* $ del *.jpg;* $ del *.htm;* $ dir [.public_html] $ lynx http://sc2a.unige.ch/~piratago

Et voualà lé trapail ! Bien entendu, vous pouvez faire un fichier batch pour tout automatiser... Le fichier batch, dans ce cas là, aura l'extension .com, et non .bat comme dans certains systèmes. UNO DOS TRES ! EN EL CULO LA METES ! Voir la section suivante pour plus d'info.

Les Fichiers Batch du VMS:

La lecture de cette section suppose que vous savez ce qu'est un fichier BATCH, par exemple sous le DOS, où ils portent l'extension

.BAT. Bon, sous VMS les fichiers batch portent l'extension .COM. Voilà la première différence. La deuxième différence vient du format interne du fichier. Sous DOS, un AUTOEXEC.BAT peut ressembler à ça:

ECHO OFF ECHO Hello DIR TIME DATE ... etc.

Sous VMS, les commandes du fichier .COM sont TOUJOURS précédées du DOLLAR $ (avez vous fait HELP DOLLAR comme recommandé au début du manuel ?). Alors, on a un truc du genre:

$ DIR $ SHOW QUOTA $ DEL *.LETTER;* $ DEL FTP_SERVER.LOG;*

Pigé ? Bon, la dernière chose que vous devez savoir est que si vous avez un fichier batch appelé OPEN.COM, pour que le VMS l'exécute, il faudra taper la commande suivante:

$ @OPEN

SANS OUBLIER LE @ ! Car si vous faites simplement

$ OPEN

VMS vous enverra un message comme quoi vous êtes très méchant et pis tout. Voilà tout. Voir la section suivante pour ENCORE plus d'info !

Le fichier batch LOGIN.COM:

C'est un fichier batch DU MEME GENRE que ceux que j'ai décrits dans la section précedente, mais qui a la particularité de s'exécuter AUTOMATIQUEMENT chaque fois que vous loggez dans le système SC2A. Par exemple, supposons que votre LOGIN.COM ressemble à ceci:

$ TYPE LOGIN.COM SHOW QUOTA PINE

Alors, chaque fois que vous loggez à SC2A...

Username: pajota Password: ******

(Divers messages de service, plus ou moins utiles)

$ SHOW QUOTA

User PAJOTA has 2344 blocks free, from 5000 allowed blocks and permitted overdrawn of...

$ PINE

...

Et voilà ! Chaque fois que vous loggez, VMS vous indique la quantité d'espace libre dont vous disposez, et vous fait sauter tout de suite après dans votre application préférée de courrier électronique ! C'est pas fabuleux ?

Quelques commandes utiles pour mettre dans le fichier LOGIN.COM:

- $ define pmdf_signature “@sys$login:signature.txt”: ajoute votre “signature file” dans tous vos mails, comme un pro;
- $ define mail_signature “signature.txt”: ajoute votre “signature file” dans vos postings sur les USENET GROUPS (les “NIOUZES”)

...à vous de créer un Signature File digne de ce nom !

Espace publicitaire:

Vous trouvez pratique ce manuel, hein?

Une blague:

Vous savez quel bruit fait un électron qui tombe ? - PLANCK ! Et quand il rote ? - BOOOHR...

Pour faire du GOPHER sur VMS:

Eh ben tapez GOPHER au dollar est promenez vous. C’est bien plus rapide que le WEB, en gros y’a les mêmes trucs mais ca plante moins souvent. Mais vous pouvez l’atteindre aussi avec Netscape.

Pour acheter des CD a travers l’Internet:

Tapez TELNET CDCONNECTION.COM, depuis le dollar ou même depuis le prompt du DOS. C’est incroyable, c’est le monstre catalogue de 8000 CD-ROM et bien plus de CD audio. Y’a même des artistes argentins, c’est vous dire s’il sont au GRAND COMPLET !!! Essayez aussi avec Netscape.

Pour gérer vos fichiers sur VMS:

Il y a une petite merveille cache dans SC2A nommée SWING. (Pourquoi SWING ???) En effet, depuis le DOLLAR tapez SWING et deux fois sur ENTREE. Vous arriverez ainsi a une sympathique interface semi-graphique sur laquelle il est teze d’effacer ou de lire des fichiers. Pourquoi les meilleures choses sont-elles cachées ???

Pour savoir s'il vous reste de l'espace disque sur VMS:

Faites SHOW QUOTA a cote du DOLLAR et vous saurez ensuite combien d'images GIF vous pourrez DOWNLOADer. Sacré verbe.

Pour faire du FTP, sur DOS ou VMS:

Depuis le prompt du DOS (en général A:\, c'est plus pratique), tapez FTP machin.truc.bidule.com, et, lorsque vous y êtes invites, logez-y avec ANONYMOUS et votre adresse e-mail comme mot de passe. Promenez vous en faisant CD nomdurepertoire, DIR et CD .. (espace deux points) pour revenir a des niveaux plus élevés. Attention avec les FANNEN[2]'s, cependant: ce sont des dangereux utilisateurs qu'y plongent le lundi a 8 heures pour en ressortir le vendredi a 17 heures...! N'oubliez pas de faire BIN avant de faire GET nanapoil.gif ou MGET *.txt au sous-repertoire /pub/mirror/misc/sexy/stories_and_photographs. Ceci vous amène dans des niveaux certainement plus bas. N'oubliez pas que pour 60 thunes vous aurez droit a une liste complète de serveurs[3] pour un public averti. Mais si vous payez, on s'en fout que vous soyez averti ou non.

Pour lire les NEWS sur VMS:

Depuis le signe du dollar $, tapez NEWS. Vous choisissez les newsgroups avec les touches de curseur et vous y accéderez avec la touche ENTREE. En effet, une étude approfondie nous a montre que l'usage de la souris n'est pas recommande: tapoter avec elle sur le clavier peut abîmer sa boule, ou vous les foutre, ce qui est plus grave. Il se trouve que la procédure de sélection du message a lire est analogue, ce qui me fait penser que, lors de la création du logiciel pour lire les news, une certaine étincelle de logique s'est allumée dans les cervelles de ces programmeurs. Faisons donc une minute de silence en leur honneur. 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60. Merci. Pour "fermer" le message, et "fermer" le newsgroup, rien de plus simple: tapez CLOSE (alors, pourquoi ne tape-t-on OPEN pour lire le fichier ?) Vous pourrez ensuite POSTer des messages, FOLLOWUPer ce qu'y sont déjà ou REPLYer directement aux auteurs vos opinions ou commentaires. Merci m'sieur Toubon pour vos idées concernant la langue Française. N'oubliez pas de taper C a cote de l'étoile pour accéder a l'éditeur sous NEWS, et

[2] fannen_aa.html
[3] weblnk_aa.html

ensuite de taper CTRL+Z et EXIT pour revenir aux NEWS. Voir plus haut pour quelques subtilités concernant les NEWS...

Espace publicitaire:

Ah qu'il est bien le manuel?

Quelques SMILEYS (sortis du livre "Smileys", par David W. Sanderson et Dale Dougherty, © 1993 O'Reilly & Associates Inc.):

Tournez l'écran 90° dans le sens des aiguilles d'une montre pour comprendre toute la subtilité de ce passage... ou bien tournez votre propre tête de 90° dans le sens contraire si vous trouvez que l'écran est bien là où il est...

Les classiques:

Standard :-) Sad :-(Winking ;-) Crying ;-(:'-(:~(Confused %-(%-) Shocked :-O 8-O Sarcastic :-] :-[Apathetic :-| Angry :-|| Happy :-D ;-D :^D Talking .^V

Caricatures:

Elvis Presley 5:-) Cheshire Cat) Cyclops 0-) Uncle Sam =|:-) Fred Flinstone 7:-) Batman B-) Felix the Cat =*0 The Enterprise =-O * * * (firing proton torpedoes)

Pour envoyer et recevoir du courrier électronique:

Depuis le $, il y a trois possibilités: soit vous voulez vous faire royalement chier et vous tapez MAIL, suite de quoi il vous faudra taper précisément, rapidement et sans faute "SEND in%<ducon@quelque.part.ailleurs>", ou bien vous utilisez EMAIL, qui vous permet d'utiliser une syntaxe plus familière, plus commode, plus élégante, du genre "SEND bordel@cettefois.tout.pres". On voit tout de suite l'avantage de la deuxième version, tellement plus conviviale. On dirait du Microsoft tout crache. Mais il y a une troisième possibilité: PINE. Mais l'auteur de ce fichier ne l'utilisant pas, vous m'en voyez strictement navré, il faudra attendre que j'apprenne à l'utiliser pour que des infos y relatives apparaissent sur ce URL. Dans les deux premiers systèmes (MAIL et EMAIL), par contre, vous taperez DIR pour voir ce qu'on vous a envoyé, vous

taperez DEL pour effacer ce que vous ne voulez pas voir et vous ferez EXTRACT fichier.extension;version pour garder ce que vous ne voulez pas effacer. Une commande très pratique est SET PERSONAL_NAME “Jeropa Jones”, car elle vous permet d’ajouter automatiquement votre nom et prénom à côté de chaque message que vous envoyez…! Vous pouvez tester votre oeuvre avec un simple SHOW PERSONAL_NAME ou bien un simple SHOW ALL, toujours depuis EMAIL, bien entendu. Voir encore plus haut pour d’autres subtilités de ce logiciel…

Pour créer des ALIAS (pour EMAIL sur VMS)

Supposez que votre ami Jean vous annonce une heureuse nouvelle. Non, pas celle-là, mais l’autre, oui, celle comme quoi il a son account sur INTERNET. Ceci est évidemment intéressant, car il faut vous mettre en contact avec lui souvent et les 10 minutes par tube aux States c’est pas donne. Et donc vous lui demandez quelle est son adresse E-MAIL et il vous dit que c’est

JEAN.PIERRE.DE.LA.CROIX@ECONOMICAL.SCIENCES.UNIVERSITY.MARYLAND.EDU.

On voit tout de suite qu’a moins d’être un dactylographe averti, le fait de taper 5 fois par jour SEND To: JEAN.PI…… peut être une tache douloureuse pour les phalanges. C’est alors qu’interviennent les ALIAS !!!! Et oui, car vous n’avez alors qu’a taper

ALIAS ADD JEAN JEAN.PIERRE.DE.LA.CROIX@ECONOMICAL.SCIENCES.UNIVERSITY.MARYLAND.ED

et maintenant vous n’avez qu’a taper SEND To: JEAN !!! OUFFF !!!! Et pour savoir quels sont les ALIAS que vous avez défini, vous n’avez qu’a taper ALIAS SHOW, ce qui donne une liste succincte et graphique vous montrant la praticité du procédé.

Pour créer des LISTES DE DISTRIBUTIONS sur VMS:

Grave souci: vous devez envoyer le même message a:

1) alsduriuagl@joaisd.dasgio.com 2) asodiutlk@lsuusdtg.aslkdaa.ch
3) siodujfgbkl@lcoisdk.vclhisug.aslgiu.edu 4) josokvjtoad@jcoislbola.vzacvag.couzsidu.ar
5) JEAN.PIERRE.DE.LA.CROIX@ECONOMICAL.SCIENCES.UNIVERSITY.MARYLAND.EDU
(et oui, lui aussi…) 6) … … … … 287237512) aslkdfoa@lsdoif.vlsiod.ashdi.net

et ceci 4324 fois par jour. Vous pouvez, bien sur, faire 287237512 fois SEND …, ou faire SEND To:…. et marquer les 287237512 adresses…, 4324 fois par jour, c’est-à-dire approximativement 1.24e12 opérations, ou 5e13 frappes au clavier par jour… Sans compter le temps et la fatigue de taper le message… (utilisez EDIT…!!!) MAIS heureusement, il existe les listes de distribution !!! C’est simple (étonnant !): il faut simplement faire CREATE LISTE.DIS (l’extension DIS est indispensable), ensuite taper:

alsduriuagl@joaisd.dasgio.com asodiutlk@lsuusdtg.aslkdaa.ch siodu-jfgbkl@lcoisdk.vclhisug.aslgiu.edu josokvjtoad@jcoislbola.vzacvag.couzsidu.ar JEAN.PIERRE.DE.LA.CROIX@ECONOMICAL.SCIENCES.UNIVERSITY.MARYLAND.EDU (et oui, lui aussi...) ... ... aslkdfoa@lsdoif.vlsiod.ashdi.net

<CTRL+Z> pour finir le fichier.

Et il ne reste plus qu'a faire, sous EMAIL, un petit

ALIAS ADD TOUS <LISTE.DIS

et le tour est joue. Dans ce contexte, SYS et LOGIN sont a remplacer selon le nom de votre account. Finalement, vous pouvez faire SEND To: TOUS et les 287237512 heureux élus recevront votre message promptement... Sacre Jean Pierre. Le monde est FOUUUUU !!!

Pour consulter l'Annuaire Téléphonique Electronique des PTT sur VMS:

Faites ATE depuis le $.

Pour consulter l'annuaire électronique de l'Université:

Faites DE depuis le dollar $.

Pour envoyer des FAX depuis le VMS:

Tout d'abord, faites EDIT nomdufax.txt et tapez votre texte en toute sécurité. Ensuite, comme il se doit, tapez FAX nomdufax.txt. TREEEEEEEEES LOGIQUE. Ensuite on vous demande si l'en-tête du fichier contient l'adresse du destinataire, le téléphone du destinataire (vous avez intérêt a le connaître...) et le titre du fax. A vous de le choisir, donc.

Pour accéder aux NewsGroups ALT depuis le VMS:

Depuis le prompt du DOS, faites TELNET GOPHER.MSU.EDU, sélectionnez ensuite GOPHER et les sous répertoires 6 ("News & Weather"), ensuite 14 ("Usenet Groups") et finalement 3 ("Items in subject order"). La vous trouverez tous les newsgroups qui malheureusement manquent a l'Uni. Entre autres, les ALT, n'est-ce pas Pierre-Etienne ? Ce qui vient d'être dit était valable il y a deux ans, alors me direz vous pourquoi ai-je laissé ceci ? Eh ben c'est très simple, passque si

jamais quelqu'un trouve un accès aux Newsgroups ALT dans un autre server, je changerais l'adresse.

Espace publicitaire:

Alors, puisqu'il est tellement bien, ce manuel, PAYEZ LES 300 SFr de registration, nomdebleu !!!

Pour critiquer Microsoft:

Il y a le serveur de Microsoft. Bill se fera un plaisir de lire vos commentaires.

Pour critiquer Jérôme Bonaldi:

Il y a CPLUSNPA@WORLDNET.NET[4] ou vous pouvez laisser les messages que vous voulez concernant Karl Zero, les Guignols ou les Deschiens.

Pour payer les 300 balles:

Ouvrez le porte-monnaie !!! Et faites votre paiement vers le compte de chèques postaux 12-3210-8. Merci !

Pour transvaser le contenu de votre account sur des disquettes:

C'est très facile. Voici un exemple clair et net(scape): Vous lisez un mail ou les news et puis vous avez soudainement envie de faire partager d'autres âmes sensibles votre engouement sur les autoroutes de l'information. Alors vous faites:

NEWS> ou EMAIL> EXTRACT connerie.txt

NEWS> ou EMAIL> EXIT

$ DIR

CONNERIE.TXT;1 NEWSRC.;1 MAIL$HFHLK004065002030.MAI;1$MAIL HDFJJRU93903490
MAIL$EUIWKFSL353349030.MAI;1$MAIL HDHZJFISU39565430.MAI;1

[4] Mailto:CPLUSNPA@WORLDNET.NET

MAIL$JFDS8743987998797.MAI;1$MAIL FDHHK847538349030.MAI;1
MAIL$HFSLKHRU847395038.MAI;1$MAIL HFJFI534590349030.MAI;1

(Taper ensuite ALT+F10, SHIFT+F: ceci met votre PC en mode serveur)

$ FTP 129.194.50.190

(ou le numéro se trouvant sur la plaquette orange collée a votre machine)

FTP> LDIR

CONNERIE.TXT;1 NEWSRC.;1 MAIL$HFHLK004065002030.MAI;1$MAIL HDFJJRU93903490
MAIL$EUIWKFSL353349030.MAI;1$MAIL HDHZJFISU39565430.MAI;1
MAIL$JFDS7543987998797.MAI;1$MAIL FDHHK847538349030.MAI;1
MAIL$HFSLKRHU847395038.MAI;1$MAIL HFJFI534590349030.MAI;1

FTP> CD A:

FTP> PUT connerie.txt;1 (ou aussi MPUT *.txt).

FTP> EXIT

$

MAIS ATTENTION !!! Il se trouve maintenant qu'une nouvelle possibilité s'est ouverte a nos accounts. Il parait maintenant que la séquence de touches

$ FTP SC2A.UNIGE.CH

marche sans problème...!!! N'oubliez pas donc de mettre votre nom et mot de passe, cela peut aider les choses... Et puis, bien évidemment, GET et PUT restent tout a fait valables dans cet environnent. On regrettera cependant le fait qu'il n'y a pas des photos avec des **gonzesses a pelos** (TELLEMENT FIN) dans le sous répertoire "[FISICETU]$UNAL" (bien que le nom nous indique une fausse piste) ou vous pouvez facilement entrer avec un simple anoniemousse. (Allé Karem, c'était une blague... Tout le monde sait que c'est chez RMORAND que ca se passe...)

Epilogue:

Simple, non ? Il faut avouer finalement que la technologie moderne nous desserve sans cesse des surprises. Je ne crois pas qu'on puisse faire mieux, malgré certains adeptes du Apple MAChin qui veulent nous faire avaler la pilule de l'interactif graphique avec air-bag, renforts latéraux, accelerator pour les couleurs a quarante degrés, isomorphisme pour la peau, et j'en passe. C'est pour cela que je m'exclame:

LONGUE VIE AU COMMAND.COM !!! (Essayez WWW.COMMAND.COM pour voir...) LONGUE VIE A L'AUTEUR !!! LONGUE VIE A CEUX QUI PAYENT LES 300 SFr !!!

Adrian Kosmaczewski

Javascript Calculator

Adrian Kosmaczewski

1997-06-18

Tato Bores en Busca De La Vereda Del Sol

Adrian Kosmaczewski

1997-09-22

Monologo N° 2000 - Domingo 9 de setiembre de 1990 - 21 horas 30 minutos.

Bueno, señores, Monologo 2000! 30 años metiendo libreto debajo de esta peluca! Mire, esta noche en lugar de hacer un monologo con lo que paso esta semana, con la actualidad, esta semana vamos a hacer un monologo recordando lo que paso los ultimos 30 años a ver si aprendemos algo, eh?

Señores: cuando alla por 1960 puse la jeta por primera vez delante de los orticones, no existia la television color, no existia Maradona, no existia el Austral - es decir, el Austral tampoco existe ahora pero es otra historia -, no existia el control remoto, no existia el yogur descremado, pero si, si existia Don Alvaro; si señores, si: Don Alvaro, el papa de la nena! Si bien Don Alvaro empezo a curtir gabinete como Ministro de Industria alla por el año '55 en la " LIBERTADORA ", que no tiene nada que ver con la Copa Libertadores, porque recien con Arturo Frondizi se convirtio en Ministro de Economia. Porque le voy a decir mas: antes de Don Arturo Frondizi no existia el Ministerio de Economia; dicen los memoriosos que para aquellos años habia un poco de guita en el tesoro y entonces con un Ministro de Hacienda tipo Serelco, alcanzaba!. Con la mishiadura aparecieron los Ministros de Economia. Lo que no queda muy bien claro es si la mishiadura trajo a los Ministros de Economia o si los Ministros de Economia trajeron la mishiadura! Lo que pasa es que hace 30 años que tenemos las dos cosas.

Por aquellos años, Don Alvaro Alsogaray se mando la famosa frase " HAY QUE PASAR EL INVIERNO ". Y pasaron y pasaron los inviernos, y las primaveras aparecieron y aparecieron - lo unico que no aparecio fue la guita -; y tambien por aquellos años '60 comenzaron lo planteos militares a Don Arturo Frondizi. En realidad el primer planteo fue el 8 de julio de 1958 pero en dos años le enchufaron 30 planteos!; y aqui con Don Alfonsin tuvimos dos planteos con los muchachos de la pomada, calcule lo que habran sido 30 planteos!. La cuestion es que los muchachos, al final, lo rajaron, y cuando el general Poggi estaba ya listo para asumir como presidente aparecio José Maria Guido - tam-

bien conocido como” JOSE DONDEMEPONGO ” -, pego un Per Saltum, entro a Tribunales, juro como presidente ante la Corte Suprema, se colo por un intersticio en una puerta de la Casa Rosada, se sento en el sillon, y cuando Poggi se dio vuelta le dijo ” ACATAA! ”

La cuestion es que Don Guido trajo a otro prohombre de la economia: Don Federico Piñedo que dijo que hay que hacer las cosas rapido y se mando en un solo dia una devaluacion del 21% y mando el dolar a la astronomica suma de 99 pesos moneda nacional de curso legal. (Chicos: si ustedes no saben lo que es eso - la moneda nacional de curso legal - preguntenle al abuelo, pero no lo hagan llorar demasiado, por favor!).

La cuestion es que Don Piñedo se las tomo ofendido por las criticas que desperto esa devaluacion y entonces aparecio de vuelta Alvaro II, que viene a ser como ” HIGHLANDER II “,” TIBURON II “,” ROCKY II “, una cosa asi.

Como el tema de ” HAY QUE PASAR EL INVIERNO ” estaba gastado, Don Alvaro invento otra cosa: invento el ” EMPRESTITO PATRIOTICO NUEVE DE JULIO ” llamado tambien ” LOS BONOS DE ALSOGARAY “. Los que se los quedaron, la verdad, se ganaron mucha guita; los que no nos los pudimos quedar, pá qué le viá contar! Es otra historia…

Mientras tanto los militares, que no tenian nada que hacer, se pusieron a jugar a los soldaditos entre ellos: hicieron una raya y dijeron: ” COLORADOS DE ESTE LADO, AZULES DE ESTE OTRO LADO, GANA EL QUE TIENE MAS TANQUES “. Nosotros, los civiles, que no teniamos arte ni parte en el asunto, porque unicamente ligabamos una bomba que nos reventara la casa, estabamos tranquilos porque tanto azules como colorados decian que todo lo hacian por el bienestar de la gente y por la salvacion de la patria; de donde se deducia que la salvacion de la patria estaba en manos del que tenia mas tanques, comprende?

La cuestion es que en el año ’63 le toco el turno de vuelta a un presidente constitucional y aparecio Don Arturo Humberto Illia, uno de los pocos Cordobeses nacidos en Pergamino que se conocen. Don Arturo Humberto Illia nombro como Ministro de Economia a Don Eugenio Blasco que muere en el cargo y entonces mi gran amigo Juan Carlos Pugliese asume como Ministro de Economia - empieza, mejor dicho, su carrera como Ministro de Economia suplente en todos los gabinetes radicales -. Pero como las cosas buenas duran poco tiempo, antes de cumplir los tres años los muchachos de la (haciendo el signo de una **insignia militar** en el hombro izquierdo con los dedos indice y mayor de la mano derecha) viñeta le dan el raje a Don Arturo Humberto Illia y designan, en elecciones limpias, y por u-na-ni-mi-dad - 3 votos - a Don Juan Carlos Ongania. El hecho de que Don Juan Carlos Ongania en la epoca del enfrentamiento entre azules y colorados haya sido azul - y legalista - y despues se convirtio en golpista - y de hecho, colorado - es porque a veces, la gente, des-ti-ñe.

La cuestion es que a Don Arturo lo rajaron porque decian que era

muy lento, que era una tortuga. Ahi tuvimos un cacho la culpa todos porque los sindicatos, la C.G.T. le tiraba tortugas en Plaza de Mayo, los medios en contra, los periodistas en contra, los humoristas le haciamos chistes - eramos una manga de boludos que pá que le viá contar -; porque el problema no era que Don Illia era lento: el problema es que los que vinieron despues fueron... fueron rapidos, y fuimos derecho pal' cara...melo, fuimos, pero bah, pero rapido!

Claro, no todo fue negrura en aquellos años porque en el '66 hubo avances: porque despues de la " NOCHE DE LOS BASTONES LARGOS " cerraron todas las facultades y entonces todos los investigadores, cientificos, matematicos, laburantes de las neuronas avanzaron: avanzaron hacia la frontera y se las tomaron y no volvieron nunca mas. Despues, aparecio algun premio Nobel que volvio: a saludar a la familia y se las volvio a tomar, total...!

Para 1969 el Ministro de Economia era Adalbert Krieger Vassena que habia mantenido el dolar mas o menos estable; pero de pronto aparecio Don Jose Maria Danigno Pastore y, como el dolar ya estaba a 350 mangos, le arranco dos ceros porque invento el peso ley 18188 - intimamente llamado " EL PESO LEY " - Don Juanca, en aquellos años - Juan Carlos Ongania - pensaba quedarse 20 o 30 años, pero aparecio el " CORDOBAZO ", el" ROSARIAZO " y el pais se movio como un " FLANAZO ". O sea que para los finales de 1970 los muchachos (haciendo de nuevo el signo de una **insignia militar** en el hombro izquierdo con los dedos indice y mayor de la mano derecha) le dieron las gracias por los servicios prestados a Don Juanca I y despues designaron en elecciones limpias y por unanimidad a Roberto Marcelo Levingston.

Roberto Marcelo Levingston es el unico presidente en toda la historia argentina desde 1810 hasta la fecha que cuando lo designaron no lo conocia ni el loro! Vea, en las redacciones, no sabian como se escribia el nombre! No habia una foto de el! Cuando, a la noche, en la sexta aparecio " LEVINGSTON PRESIDENTE " la gente preguntaba " PERO EL PRESIDENTE DE QUE PAIS SERA ESTE BUEN SEÑOR? " Y porque para colmo, cuando lo designaron el estaba en la Junta Interamericana de Defensa en Washington! Asi que aqui estabamos como los indios que se golpean el codo: en bolas, y a los gritos!

Por fin, Don Levingston aparecio y dijo " SOY EL PRESIDENTE " y se sento en el sillon a esperar ordenes. Lo que pasa el problema fue que mientras estaba esperando las ordenes empezo a jugar un jueguito que decia: " PESE A TODO, YO SOY EL PRESIDENTE ". Don Lanusse, que era el inmediato superior, no le gusto nada la cosa, pero roce va roce viene Don Levingston lo destituye a Lanusse, Lanusse escucha eso, caza el tubo y lo destituye a Levingston, y como donde manda Teniente General no manda General de Brigada Levingston volvio rapidamente al anonimato. Cansado ya de hechar presidentes - habia echado dos - Don Lanusse penso:" PARA PENSAR COMO YO, NADIE COMO YO ". Entonces agarro y se nombro presidente sin dejar el cargo de Comandante en Jefe. Astuto el hombre! Y enseguida invento

una cosa que se llamo el G.A.N.:" GRAN ACUERDO NACIONAL ". Y lo mando al Coronel Cornicelli a verlo a" PUERTA DE HIERRO " a mi gran amigo Juan Carlos Can... Juan Carlos no (risas), Juan Domingo, Juan Domingo (aplausos), Juan Domingo Cangallo y le dijo que si entraba en el G.A.N. le devolvia todos los sueldos del '55 hasta la fecha. El viejo dijo " LO PRIMERO ES LO PRIMERO ", cazo la mosca, lo dejo al gobierno con el G.A.N. y con las ganas. Y entonces Don Lanusse se chivo y se mando la famosa frase que" EL VIEJO NO VOLVIA PORQUE NO LE DABA EL CUERO ". Pero como el viejo debajo de las arrugas todavia le quedaba un cacho de quiero... de cuero, volvio para mostrarlo en vivo y en directo y formo un frente civico que se llamo" FRE.CI.LI.NA. ". Pero como la Frecilina tenia nombre de antibiotico lo cambiaron por" FRE.JU.LI. ". Escuche, Frecilina, Frejuli, Frejupo, son todos remedios del mismo laboratorio! Vienen en pildoras, en inyectables, en supositorio, uselo como le de las ganas!

La cuestion es que en aquellos años '73 aparecio " LA NUEVA FUERZA ", un partido politico inventado por mi gran amigo Alsogaray que tenia como candidato a presidente a mi gran amigo Julio Chamizo, el que quiere acordarse, que se acuerde! La cuestion es que el 25 de mayo de 1973 asumio el tio, no este, otro tio, el tio, el tio Hector J. Campora, y como el eslogan era" CAMPORA AL GOBIERNO, PERON AL PODER ", los muchachos del bombo rapidamente renunciaron a don Hector. Renuncio el presidente, renuncio el vicepresidente, renuncio el presidente provisorio del Senado Diaz Vialisi, una cosa ahi, que se yo lo que hicieron. La cuestion es que quedo como candidato a Presidente de la Republica el presidente de la camara de Diputados Raul Lastiri, que casualmente era yerno de Lopez Rega!. La cuestion es que Lastiri - conocido tambien como" JOSE CORBATA " porque tenia un monton y le encantaban - llamo a elecciones y gano por unanimidad la formula " MENEM-MENEM... ", digo, no, la formula" PERON-PERON ". Peron se muere y de estar mal pasamos a estar peor porque viene Isabelita y lo trae a Celestino Rodrigo que se manda el famoso" RODRIGAZO " que nos deja a todos con el tuje pal' norte! La moral de la historieta es que Don Celestino, que yo sepa cabe destacar, y que yo sepa, fue el unico Ministro de Economia, que se comio canas por cuestion de su gestion como ministro, cosa que no le ha pasado a ningun otro ministro de economia, nunca mas, se han salvado todos, la verdad es que es un misterio, que no se por que! (a su libretista) Como seguia esto? (el libretista le sopla, y Tato sigue). Ah!, si.

Despues de Celestino Rodrigo, despues de Celestino Rodrigo aparecio Tony Cafiero, si, si,si, Tony Cafiero, el del " SI ", el del" SI LO HUBIERA SABIDO NO LLAMABA A PLESBICITO "! Y despues de el aparecio Mondelli - que Isabel decia" NO ME LO TOQUEN AL GORDITO " -.

Cuando se murio Peron - es una acotacion que le voy a hacer yo - estaba laburando en este canal, me llamaron para decirme " VAMOS A PARAR UN POCO CON LOS PROGRAMAS HUMORISTICOS, HAY QUE HACER DUELO ", y yo pense que estaba bien para que lo suspendan un par de semanas (silencio durante algunos segundos, y luego

risas)... La verdad es que no lo suspendieron un par de semanas, lo suspendieron un par de años! Porque despues vinieron los muchachos del '76 de vuelta y la siguieron... Porque en aquel entonces eran largos los duelos, comprende?!

Y asi llegamos, a la epoca del proceso, de los Ministros de Economia, era Jose Alfredo Martinez de Hoz, y el proceso lo voy a pasar por alto porque, la verdad que, no, mejor no recordarlo, cierto?

Por eso hice un Per Saltum y apareci en la democracia, en 1983, con Alfonsin, Grinspun, Sourrouille, el Austral, el desagio, Juan Carlos Pugliese II, el bolonki, y Jesus Rodriguez casi como Jesus termina crucificado. Mientras en estos tiempos la hiperinflacion y los empresarios le apretaban el gañote a Don Raul Alfonsin, aparecio Carlos Saul I, primer presidente electo que decia que tenia el equipo formado, listo para salir a la cancha y ganar por goleada! Don Raul, que queria quedarse 6 años, ni un dia antes, ni un dia despues, no le quedo mas remedio que tirar la esponja y de paso le tiro el gobierno por la cabeza a la patilla mas gorda de America, Carlos Saul I.

Y aqui estamos señor. 30 años. 30 años bancandose 16 presidentes y 37 Ministros de Economia que se la pasaron diciendo " ESTA ES LA CRISIS MAS GRANDE QUE ESTA SUFRIENDO EL PAIS "," HAY QUE REDUCIR EL GASTO PUBLICO "," HAY QUE LABURAR MAS "," HAY QUE INVERTIR EN EL ISPA ". Mientras tanto, quiere que le diga una cosa?, mire, este peso moneda nacional (sosteniendo el billete de un $mn en la mano, con otros billetes –un $ley 18188, un $argentino, un Austral– sobre la mesa) le arrancaron dos ceros por este otro peso ley 18188; a este le arrancaron cuatro ceros por este otro peso argentino, y como si esto fuera poco le sacaron tres ceros mas por este peso... por este Austral. O sea que extirparon, le extirparon nueve ceros a este pesito de aca delante. Y como este Austral equivale a mil millones de pesos moneda nacional, y como en aquel entonces se compraba con 83 $mn un dolar, este Austral equivale a DOCE MILLONES DE DOLARES... (risas, mezcladas con silencio, lagrimas e ironia), lo cual parece un chiste, si no fuera una joda grande como una casa... Y yo todavia (aplausos), yo todavia tengo confianza, tengo confianza, por eso le digo a los politicos y a los funcionarios - no a todos los politicos ni a todos los funcionarios porque hay que preservar las instituciones - algunos politicos y algunos funcionarios que estan ahi viendome, si siguen haciendo las cosas que estan haciendo yo voy a tratar de estar aca todo el tiempo posible para seguir jodiendo! Y para cuidarlos tambien... Y para preservarlos de la maquina de cortar boludos; porque si pusieramos la maquina de cortar boludos dentro de la maquina del tunel del tiempo, y se pusiera a cortar boludos historicos con retroactividad... otra hubiera sido la historieta hoy! Historieta que como pais, no creo que nos merezcamos - esto lo dice mi libretista Santiago Varela... yo... no estoy tan seguro! Un cacho de culpa tenemos tambien...! -.

Por eso les digo, mis queridos chichipios, seguir laburando, vermouth con papas fritas, y... (aplaudiendo dos veces, levantandose y termi-

nando el monologo como todos los domingos) GOOD SHOW!!!

The JAVA© Programming Language

Adrian Kosmaczewski

1997-12-11

Since I read a news article about Java©, on August '97, I wanted to know more about programming for the World Wide Web. I tried first JavaScript©, but its many features made me... buy a book about JAVA©... and I love it! It's a really great language. Not so easy, but thrilling! My first applets for JAVA© are in the following links... they are almost all under heavy construction, so don't get angry with me! I'm just beginning with all this!

- A VERY SIMPLE password reader[1];
- A simple calculator[2] in JavaScript;
- The same calculator[3], but written in Java (I love calculators, as you can see !);
- The MegaSearch page[4], written with JavaScript;

Further Information about JAVA©:

If you want to know more about it, two books can save your life:

- JAVA Programming for Dummies, © 1996 IDG Books Worldwide, Inc.
- Teach yourself JAVA 1.1 in 21 Days, © 1997 by Sams.net Publishing

And you can also take a look to these WWW Sites:

- Sun Microsystems[5];
- JavaSoft[6];
- Gamelan[7];
- Java World[8];

[1] javapass_aa.html
[2] /blog/javascript-calculator/
[3] javacalc_aa.html
[4] megasearch_aa.html
[5] https://web.archive.org/web/19971210195754/http://java.sun.com/
[6] https://web.archive.org/web/19971210175409/http://www.javasoft.com/
[7] https://web.archive.org/web/19971014182655/http://www.gamelan.com/
[8] https://web.archive.org/web/19971210175814/http://www.javaworld.com/

- Java Report[9];
- Java Applet Rating Service[10].

JAVA© is a registered trademark of Sun Microsystems

Update, 2023-07-23: 25 years after I published this web page, Java applets are deprecated and forgotten… well, almost: you can run them again on Chrome if you want to, thanks to WebAssembly. Here's how[11].

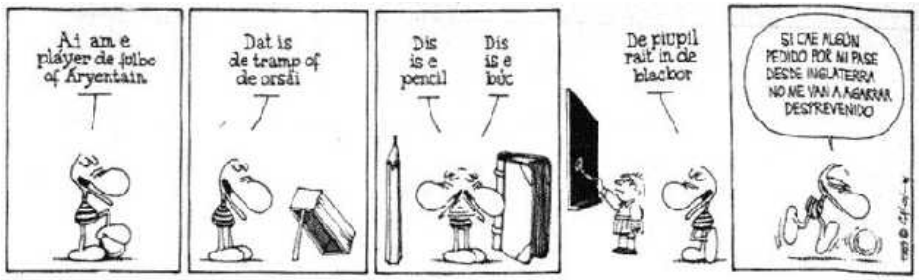
[9] https://web.archive.org/web/19971210205943/http://www.sigs.com/
[10] https://web.archive.org/web/19971210175017/http://www.jars.com/
[11] https://akos.ma/blog/java-applets-in-2023/

My Favourite Comic Strips

Adrian Kosmaczewski

1997-12-19

Un Clemente del año 1977:



Otro Clemente pero del año 1997:

Calvin & Hobbes en français:

My Job at FIS International Co. Ltd.

Adrian Kosmaczewski

1998-02-24

[1]

On october 1997 I joined the team who made possible the publishing of one of the largest databases about fishing, aquaculture, seafood processing and vessels' trading on the Internet, the Fish Info Service that can be reached with any browser at www.sea-world.com[2].

FIS concentrates in one single site Company Information, an electronic Trading Market, worldwide Market Prices and Reports, Hot News about fishing, aquaculture and seafood processing, the complete list of Worldwide Shows and Fairs, weather links, and lots of useful information.

Since I began, I wrote many pages for companies publishing on the site. You can see them at the following addresses (some are not yet available, because still in construction):

- Invertec Seafoods[3]

[1] http://www.sea-world.com
[2] http://www.sea-world.com
[3] http://www.sea-world.com/invertec

- Estudio de abogados Muñiz-Ziches, Forsyth, Pérez-Taiman & Luna-Victoria[4]
- The server's error page[5], and some other pages and CGI for our intranet...
- Frio Sud Nor S.A.[6]
- South American Cargo S.A[7]
- El Dorado del Mar[8]
- Azcona Frigomarina[9]
- Complejo Ransa[10]
- Refrigerados Iny[11]
- Biotac[12]
- La Isolana[13]
- Legaspi Products[14]
- Pesquera San Antonio[15]
- TYCSA Fishing Wire Ropes
- Interaliment
- Fujian Provincial Aquatic Products Trade Co.
- Hunan Xinhai Fishing Goods Co., Ltd.
- Haikui Aquatic Products

There's another page that I formatted to HTML on the Internet, it's a lecture on Policy for the Attainment of Sustained Economic Growth[16] written by Dr. Carlos Sabillon, from the University of Geneva.

But out of that work stuff, I really enjoy triying new techniques on my pages; I am actually improving my Java, JavaScript and VBScript programming skills[17], as well as my knowledge on Active Server Pages, a new technology for creating CGI scripts for servers running on Windows NT 4 Server, with the Internet Information Server extensions installed. For example, check out this game[18] written with Active Server Pages !

[4] http://www.sea-world.com/munizlaw
[5] http://www.sea-world.com/errorpageplease
[6] http://www.sea-world.com/sudnor
[7] http://www.sea-world.com/sac
[8] http://www.sea-world.com/eldorado
[9] http://www.sea-world.com/azcona
[10] http://www.sea-world.com/ransa
[11] http://www.sea-world.com/iny
[12] http://www.sea-world.com/biotac
[13] http://www.sea-world.com/laisolana
[14] http://www.sea-world.com/legaspi
[15] http://www.sea-world.com/sanantonio
[16] https://web.archive.org/web/20001108235800/http://uni2a.unige.ch/~barras3/
[17] /blog/the-java-programming-language/
[18] http://www.sea-world.com/fis/dami/aspgame.asp

New Websites

Adrian Kosmaczewski

2002-07-13

Back in Switzerland, I've resumed my activity of website design and development. Here are some websites I recently helped put online, check them out:

- Martin Ennals Award[1]
- Job Sélection[2]
- Quorum Management[3]

And next Monday I'll be starting work as a software developer at Soft-Plumbers[4] in Geneva, Switzerland!

[1] https://web.archive.org/web/20020925051640/https://www.martinennalsaward.org/

[2] https://web.archive.org/web/20021207085824/http://www.job-selection.ch/fr/accueil/

[3] https://web.archive.org/web/20030726144524/http://www.quorum-management.ch/

[4] https://web.archive.org/web/20020603033125/http://www.softplumbers.com/

Palais Des Nations Unies

Adrian Kosmaczewski

2004-04-08

Desde que llegue a Ginebra en 1991, tuve varias veces la ocasion de entrar en el palacio de las Naciones Unidas de Ginebra. La primera vez fue con Lucia, alla por marzo del 91, cuando vino de visita con sus viejos, ya que ellos se habian mudado al sur de Francia mas o menos para la misma epoca. La cosa fue que entramos al palacio, creo que fue un domingo, y paseamos por los diferentes salones, visitando esa inmensa estructura, que es el edificio que sirve de sede a la sucursal europea de la World Company.

Es un edificio impresionante, sinuoso, en el que no es dificil perderse. Esta construido sobre una loma que da al lago, asi que no es poco comun entrar al edificio y darse cuenta que uno esta en el tercer piso, y no en la planta baja como suele suceder en otros edificios. Ni tampoco es inhabitual que para ir al ala este del edificio haya que subir dos pisos, atravesar tres pasillos sinuosos llenos de puertas cerradas una al lado de la otra, para luego tomar el ascensor 15 y salir del edificio por la puerta 34. Originalmente construido para la Sociedad de las Naciones, la ONU tomo posesion de los lugares luego de que la primera se fundiera.

Recuerdo haber hecho el comentario, una vez que me perdi en esos pasillos, a unas personas que estaban en algun ascensor, de que el edificio es tan tortuoso como la politica internacional, y creo que aun se estan riendo tanto les parecio gracioso mi comentario. No lo dije en broma. Es mas, perderme en aquel edificio no me hizo ninguna gracia. Los pasillos son bastante tristes, lugubres, la arquitectura es al mismo tiempo monumental pero muy pesante, con tantas puertas cerradas, con plaquetitas al lado de las mismas, con gente que va y viene en silencio, con papeles acumulados por todos lados. Y un omnipresente olor a encierro, a burocracia. Paradojicamente, o tal vez no tanto, los jardines que rodean el edificio son de una rara belleza, y en esta epoca de primavera estan poblados de una vida exhuberante. El contraste es profundo y pesante. Hay incluso una familia de pavos reales (regalados por alguna familia real en agradecimiento de algun favor politico) que se ponen a cantar invariablemente cada atardecer a la misma hora.

Luego de aquella vez, no volvi a pisar el palacio de las Naciones hasta el año pasado, 2003. Entre tiempo paso de todo, y para resumir, el

asunto fue que empece a trabajar como webmaster benevolo para una fundacion que cada año otorga un premio a un defensor de los derechos humanos. Y cada año se hace una ceremonia en el ultimo piso del palacio, que es un inmenso salon con un balcon que tiene una vista hermosa hacia el lago Leman, los Alpes y la ciudad de Ginebra.

A fines del 2003 me toco volver a ir, pero esta vez como profe; durante mi periodo de desempleo del año pasado encontre un curro (tanto en el sentido argentino como español de la palabra) como profe de informatica y me toco ir a dar unas clases de SQL Server 2000 en la ONU, para gente de la Comision Economica para Europa, mas precisamente.

Y ayer fui otra vez, y de lejos, fue la mas divertida de todas. Esta es la anecdota.

Resulta que tengo una primita de origen suizo-aleman, con la cual hablo siempre en ingles, ya que mi aleman y su frances no dan para una charla extendida. La cosa es que despues de varios años en Estados Unidos, la flaca decidio volver a vivir en Suiza, y se vino a Ginebra ayer para un par de entrevistas. Y charlando con ella le pregunto, que te gustaria hacer? Y me responde: me interesaria un trabajo en el campo de los derechos humanos. Y entonces se me ocurrio, y le propuse llevarla a la ONU, ya que ayer era tambien el dia de la entrega del premio ese, version 2004, en el mismo lugar del año pasado. La ocasion ideal.

La idea le gusto, pero obviamente se impone un problema basico: uno no entra en la ONU asi como asi; se imaginaran que los controles de seguridad son estrictos, numerosos y jodidos; y si no estas en la lista de invitados (que era lo que me permitia entrar anoche pero que representaba un problema para ella) dificil que te dejen entrar.

Pero uno no es caradura por nada. Con mi primita nos presentamos en la entrada de las Naciones Unidas, la que queda enfrente de la sede de la Cruz Roja, y le digo a la mina de seguridad: venimos por la ceremonia del premio, estamos en la lista de invitados. Y le doy mi cedula de identidad. Mira el apellido polaco, lo encuentra en la lista, y mirandola a mi primita me pregunta: “Estan juntos?”, a lo cual le respondo con un categorico “Claro!”. Mi primita, pobre, suiza de lo mas reglamentaria, me miro con ojos desorbitados mientras pasabamos el detector de metales, yo lo mas impavido del mundo, bromeando con el chabon de la seguridad, y tomandola por la cintura a mi acompañante mientras caminabamos por los jardines del palacio.

La flaca no podia creerlo, y me dice: “Le mentiste!” a lo cual mi respuesta fue tambien contundente: “Obvio! No esperabas menos de un primo argentino, no?”.

Lo mas comico es que despues nos enteramos que uno de los invitados de la ceremonia, un alto dignatario de no se que organizacion o pais, estaba bloqueado en la entrada (la misma que habiamos pasado) simplemente porque su nombre no estaba en la lista. La mire

a mi primita y le dije, tendria que haber entrado con nosotros, no?

Escrito en Un Bar

Adrian Kosmaczewski

2004-11-04

escrito en un bar, en la esquina de alvarez jonte y avenida san martin, barrio de la paternal, ciudad de buenos aires.

jueves 4 de noviembre del 2004, 1730 horas.

48 horas antes de enchufarme en la matriz de vuelta, un tostado y una pepsi me ayudan a hacer tiempo

veo colectivos, un 24, un 146 diferencial, un 105. un patrullero, camiones pesados, claro, por la avenida san martin transitan camiones de carga. es la “red de transporte pesado” de la capital, que le dicen.

en la mesa de al lado, un hombre de unos 80 años, con un parkinson muy avanzado, lee el diario popular.

afuera esta nublado, frio.

un cliente entra, pide un cortado. el mozo lo saluda, mientras en la barra un flaco lava vasos, los seca, los deja listos para que otros como uno se sienten un rato a pasar el tiempo.

en la esquina de enfrente, la pizzeria “yatasto”, y enfrente de esta, la ferreteria de “julio y dora”, presente en el barrio desde 1950. en la puerta, un señor que debe ser julio, y en las persianas metalicas, publicidades viejas, que me hacen pensar en los lubricantes bardahl, en las pilas varta, en el shampoo valet, en el chocolatin jack.

besos en otra mesa; primavera fresca, lluviosa, pero primavera al fin.

el anciano de la mesa de al lado se levanta dificilmente, le cuesta mucho caminar. me mira con ojos profundos, lejanos, doloridos. desvio mi mirada, tal vez por pudor, dolor, vergüenza.

la pareja de enamorados le pide al mozo una lagrima y tostadas con dulce de leche y manteca; dificil no escucharlo, la flaca tiene una voz aguda y no me cuesta oirla claramente.

suena un telefono. otro 24 se detiene en la esquina, baja una señora con un bebe.

me termino el tostado, y se va haciendo hora de terminar de escribir.

se va haciendo hora de irse, como sucede siempre. para luego volver, como sucede siempre.

Justo Antes De Irme

Adrian Kosmaczewski

2004-11-06

Buenos Aires me despidió con un día espléndido. Estoy sentado delante de la puerta 8 del aeropuerto de Ezeiza, con los ravioles de la abuela en mi estómago y la imagen de ella y mi viejo alejándose, después del último abrazo.

Escena cuantas veces repetida, hasta el cansancio, como si fuese un acto fallido de mi karma que se repite una y otra vez; como si la vida estuviese insistiendo para que yo aprenda algo de todos estos encuentros y desencuentros. La imagen de Condarco 2514, Sarmiento 1287, Albarellos 1548, Debenedetti 1318, Libertador 1574, tantos lugares, tanta gente, tanta vida que va y viene, el acto maravilloso de la existencia y de la plenitud.

Tengo al lado de mi compu, en la mochila, una quena. Hecha a mano, se transformó ipso facto en un objeto de culto, como la representación más tangible de un cariño que ya no tiene más necesidad de expresarse con palabras. Ayer, mis primeras notas, en dúo con un ser entrañable que la vida me puso en el camino, y que sin ser hermano, se transformó en uno.

Cosa rara del destino, la mochila en que llevo la compu resultó tener un bolsillo externo perfectamente adaptado para llevar la quena conmigo, de un lado a otro. Algo me dice que una cosa especial está naciendo entre ella y yo. Mi primer instrumento musical; es difícil describirlo.

Resuenan en mí las risas de anoche, el vino compartido y que acerca, la música que une e identifica. El abrazo fraterno que alimenta el alma, que quiere decir tanto pero sin palabras. Si, un par de lágrimas.

Venir a Argentina es siempre emotivo, hermoso, simple, alentador, violento, intenso. No es gris: es blancos y negros, rabiosos, fuertes, casi obscenos.

Y “Ay Ay Ay” en mis oídos; el sonido de los Piojos, el recuerdo de lo pasado y el anhelo de lo que vendrá. Ansiedad? Y si, un poco; como siempre, como cada vez.

Y en la pista veo a los chabones vaciando el Airbus A340 de Lan Chile que acaba de llegar; una vez estuve allí. No hace mucho, nada es

hace tanto, ni nada está tan lejos; la memoria es una realidad completa, pero que solo una persona vive a la vez. Perderla seria negarse.

A Proposito Del Coso 50: Tripode Plastico De La Pizza en Forma De Mesita

Adrian Kosmaczewski

2004-12-05

Estimado Sr. Podeti[1],

Ante todo, reciba usted mi mas sincero “aguante el aguante” ya que lo que usted hace es tremendo. Espero que esta misera, deleznable anque irrelevante contribucion sirva para, al menos, replantearse algo. No se, algo. Lo que usted quiera. Vamos, hombre, no tenga miedo.

Con respecto al coso numero 50[2], el coso de tres patas que puede ser confundido con una mesita o con un tripode falluto, y que sirve para proteger la musarela del carton, porque de otra manera se estropea el carton, le dire que yo de pibe, me inspiraba en la imagen siguiente, y cuando no quedaba mas pizza ni carton, me quedaba horas jugando al Sputnik.

El parecido es, reconozcamoslo, asombroso. Asi nomas. Si bien el Sputnik tiene una pata de mas que el coso numero 50, yo jugaba durante horas al satelite inutil, porque que cosa mas inutil que el Sputnik? Acaso lo unico que hacia era “bip, bip, bip” (lo cual puede ser la onomatopeya del Sputnik para su coleccion de onomatopeyas ineditas), aunque tambien sirvio para enfriar aun mas la ya gelida guerra fria. Yo le encontre un uso mas relevante, heroico, que describire un par de lineas mas abajo.

Lo de la onomatopeya del Sputnik es por demas interesante, porque en el espacio, donde no hay aire, no puede haber sonido; entonces el “bip, bip, bip” se escuchaba en realidad en la base de Baikonur, en el Kazajstan. Es decir, que este es un detalle que se le escapo a George Lucas, porque imaginese que cuando el Halcon Milenario se acerca a la Estrella de la Muerte, no solamente no se ven Sputniks, sino cazas “TIE” que hacen un ruido asi: “Uaaaaaaaaaaaa” (otra onomatopeya, similar a las de las espadas laser[3], lease de preferencia con voz carrasposa).

[1] https://web.archive.org/web/20041208120517/http://weblogs.clarin.com/podeti/

[2] https://web.archive.org/web/20111122180229/http://weblogs.clarin.com/podeti/2004/11/30/coso_50_trpode_plstico_de_la_pizza_en_forma_de_mesita/

[3] http://weblogs.clarin.com/podeti/2004/12/03/Â¡ah_ahora_resulta_que_como_soy_sordo_no_me_puedo_quejar_de_ese_ruido_infernal/

Cabe destacar, que “TIE” segun el libro para nerds de la Guerra de las Galaxias, es “Twin Ion Engine” es decir, que el “Uaaaaaaaaaa” corresponde al ruido de dos motores ionicos gemelos, por eso hay que poner vos carrasposa, es decir, ni siquiera se si esta ultima palabra existe en el diccionario, pero describe bien el artilugio vocal necesario para ir a trabajar como sonidista en Joligud.

Otro dia le mando la onomatopeya de Chewbacca, que es algo como “Wooooooooo”, es decir, similar al ruido de un caza TIE pero con voz de baritono resfriado.

Y entonces, uno deduce que:

1. El campo magnetico que retiene al Halcon Milenario en la Estrella de la Muerte tambien mantiene cercana la atmosfera de la recientemente destruida Alderaan (para los que no vieron la pelicula, que les puedo decir, no existen). Esa atmosfera “robada” permite que se escuchen sonidos, y por eso se escucha el caza que pasa en vuelo rasante sobre el Halcon Milenario, mientras que Han Solo se pregunta con Luke Skywalker si el caza (que, es sabido, no posee hiperpropulsor), formaba parte de un convoy y se perdio, o se dirige al satelite que se encuentra mas alla.
2. El Sputnik debia tener las antenas apuntando para atras como un cometa porque a) iba muy rapido o b) en realidad nunca salio de la atmosfera o c) los que diseñaron la capsula eran fanaticos de la grande mitad de musa, jamon y morrones, mitad roquefort. Con dos fainas, por favor.
3. En los años 80 compraba pizza en la pizzeria “Halloween” de Libertador esquina San Martin de Vicente Lopez (es decir, San Martin esquina San Martin). En esa pizzeria, Ariel (vaya un saludo) nos vendia la pizza en una caja de carton redonda, que cuando se puso de moda la serie “V Invasion Extraterrestre” yo usaba (cuando no tenia mucha musa pegada) como el ovni que cubre la ciudad de Los Angeles. O sea, que combatia como Donovan contra los extraterrestres, pero con el Sputnik. Era una lucha tremenda. Debo admitir que los malditos reptiles ganaban seguido (los muy turros disparaban carozos de aceituna) pero a veces aparecia el lagarto bueno (como se llamaba, Willy?) y decia cualquier gansada (hablaba mal en español porque lo iban a mandar a Arabia, el que tenga memoria, que recuerde, y el que no, que tome Memorex, como en las historietas de Quino).
4. Si Kennedy hubiese sido ruso, y en vez de Apolo 13 hubiesemos tenido que ver a Tom Hanks en “Soyuz 13”, relatando la historia de como casi se pierden en el espacio nuestros tovarich al querer ir a la Luna, la frase seria: “Байконур, у нас проблема.” Y en vez de usar el modulo “Eagle” de tres patas, usarian, claro esta, un Sputnik.

Sin mas, saludo a Ud. atte., deseandole que la Fuerza este con usted, siempre,

Adri

PD: la onomatopeya de Robin Williams en “Mork y Mindy” es “Nenonenonenonenoneno” (o me estoy confundiendo de serie? Ah, no, ese es el ruidito del robot que acompañaba a Gil Gerard cuando hacia de Buck Rogers en la serie homonima en los 70).

Asteroid

Adrian Kosmaczewski

2005-01-03

It turns out that there is an asteroid (number 15609) named “Kosmaczewski” (well, that won’t surprise many of those who know me too well…) Here’s the story of how it happened, taken from this document:

> SARA’S ASTEROID. A science project that looked at the way pesticides affected animals like worms and slugs has earned Hamden ninth grader Sara Kosmaczewski the honor of having an asteroid named after her. The youngster, along with her teacher, participated in the third annual Discovery Young Scientist Challenge, a national science contest; her project, which involved testing the animals with pesticides similar to those used to combat West Nile Disease, found that the amount of the chemical sprayed was more important than the particular type. Fewer than 10,000 people have asteroids named after them; scientists at the Massachusetts Institute of Technology (MIT) suggested naming asteroids after Sara, the other contest participants, and their teachers.

If you ever read this, Sara, the Kosmaczewski family owes you something absolutely incredible!

Cuanto?

Adrian Kosmaczewski

2005-01-15

Al principio fue la hoja A4, que digo, la hoja oficio, que digo, el papiro.

- La palabra “bit”, que quiere decir “pedazo” en ingles, se usa como acronimo de “binary digit”, o digito binario: un bit puede tener dos valores, solamente: 0 o 1. Cualquier interruptor electrico muestra claramente como se “representan” los bits en electronica.
- Un byte es una secuencia ordenada de 8 bits. Los franceses no llaman “byte” a esa secuencia, sino “octet”.
- Cualquier letra de cualquier alfabeto occidental, ya sea cirilico, griego o latin, se puede representar con solamente un byte. Con 8 bits se pueden representar 255 letras distintas. Pero no se pueden representar todas simultaneamente; no todas las secuencias de 8 bits quieren decir lo mismo, sino que hay que ponerse de acuerdo con el idioma que se use.
- 1024 bytes es un Kilobyte (KB); si bien, tradicionalmente, el prefijo “Kilo-” significa 1000, 1024 es igual a un 2 elevado a la potencia de 10. Luego, los ingenieros electronicos, por extension, llamaron Kilobyte a una cantidad de 1024 bytes, ya que es tecnicamente mas apropiado y coherente poner una potencia de dos de componentes en un circuito que una potencia de diez.
- Es conocida la cita atribuida (falsamente) a Bill Gates segun la cual dificilmente alguien necesitaria mas de 640 KB de RAM en su computadora. Pero Bill Gates ha dicho muchas otras cosas[1] y es facil confundirse.
- Un diskette 3.5’’, el clasico de plastico duro inventado por Sony y popularizado por la primera Apple Macintosh en 1984, puede guardar 1.3 Megabytes (MB) de datos, unos 1300 KB. Con eso alcanzaba, hace unos 15 años, para guardar un par de archivos WordPerfect[2], un par de hojas de calculo Lotus 123[3] y alguna cancion en formato MIDI.
- Un Megabyte (MB) son 1024 KB.

[1] http://en.wikiquote.org/wiki/Bill_Gates

[2] http://www.wordperfect.com

[3] http://lotus.com/products/product2.nsf/wdocs/123home

- El Windows 3.1 (1992) venia en 7 diskettes de ese tipo, de los cuales se usaban solo 6 para la instalacion, ya que el septimo traia drivers para impresoras no estandares y esas cosas. El Word 2.0 venia en 2 diskettes en el 92, y el Excel 5.0 venia en 4 diskettes en el 93.
- La conexion a Internet que tengo en casa me permite bajar 100 MB en 17 minutos.
- Un CDROM puede guardar 650 MB, es decir, 500 diskettes 3.5''.
- El Microsoft Office 2000 venia en dos CDROM.
- Un DVD puede guardar 4.7 Gigabytes (GB) de datos, mas o menos 2 horas de video en alta resolucion, es decir, unos 7 CDROM.
- Un Gigabyte (GB) son 1024 MB. Siempre vamos de a 1024, como habran notado.
- El "Visual Studio.NET" de Microsoft viene en 1 DVD o 5 CDROM (incluye la libreria MSDN o Microsoft Developer Network Library).
- El otro dia baje 1.5 GB de informacion en formato PDF del sitio developer.apple.com[4] y tardo unas 5 horas en bajar.
- El disco rigido de mi compu es de 250 GB, unos 50 DVD.
- 4 discos rigidos de 250 GB hacen 1 Terabyte (TB) de datos; hoy dia se empiezan a vender discos rigidos con esa capacidad a precios razonables. De aca a 1 año y medio seran estandar.
- Un Terabyte (TB) son tantos GB, bueno, ya saben, no?
- Se estima que la biblioteca del Congreso de los Estados Unidos, segun parece una de las mas grandes del mundo, tiene unos 20 TB de texto.
- El "Internet Archive" en http://www.archive.org/, que es un web-site que mantiene registro automatico de todo lo que aparecio en Internet alguna vez, tiene un Petabyte (PB) de informacion. Un Petabyte son 1024 Terabytes, por si quedaba alguna duda.
- Un Exabyte, son 1024 Petabytes (cuando no), es decir, 1 millon y pico de Terabytes.
- El procesador G5[5], que es el "cerebro" de la computadora que tengo en casa, es capaz de manejar un maximo de 16 Exabytes de memoria viva (RAM) simultaneamente, porque es un procesador "64 bits". (el Pentium de Intel[6] es un procesador de "32 bits", y por lo tanto puede procesar un maximo de "solo" 4 GB de RAM simultaneamente); obviamente que aun no se venden chips RAM de tal capacidad; lo mas que se encuentra, a precios

[4] http://developer.apple.com
[5] http://www.apple.com/g5processor/
[6] http://www.intel.com/products/desktop/processors/pentium4/

aun exorbitantes, son chips de RAM de 2 GB (actualmente a unos 2500 dolares por pieza).

- Mi computadora tiene dos procesadores G5 funcionando en paralelo, efectuando 2 mil millones de instrucciones por segundo cada uno (algo asi, mas o menos). De todas ellas, rara vez alguna es fundamentalmente importante: la mayoria sirven para mover la flecha del mouse o activar el screensaver al cabo de 5 minutos de inactividad.
- 16 Exabytes es el equivalente de 16000 Internet Archives, u 800.000 Bibliotecas del Congreso, o 3.400 millones de DVD, o 23.800 millones de CDROM, o 12 billones de diskettes 3.5''.
- En el planeta somos solo 6.500 piojosos millones de infelices, de los cuales el 90% trabaja para subvencionar el bienestar del 10% restante. Solamente 800 mil personas tienen acceso a Internet, a una velocidad promedio de 40 kbps (Kilobits por segundo, es decir, 5 kilobytes por segundo).
- Las velocidades de transmision se miden en Kilobits por segundo en vez de Kilobytes por segundo por una razon muy simple: queda mejor decir que tu modem de mierda va a 40 que a 5 lo-que-sea por segundo. Vende mas, simplemente.
- Con mi conexion se tardarian 6 millones de años para bajar 16 Exabytes, pero con un modem de 28 Kbps (estandar hace 15 años) se hubiesen tardado 450 millones de años. De seguir esta progresion (que seguramente se acelerara, no cabe duda), dentro de 15 años se tardara solamente 80 mil años, y dentro de 30 solamente 1000 años. Dentro de 45 años, solamente 14. Dentro de 60 años, se podran bajar 16 Exabytes en unos 68 dias. En ese entonces tendremos 90 pirulos, muchos de entre nosotros ya no tendremos un estado mental saludable como para digerir tales numeros, y muy probablemente, ya ni nos interese.

Vivimos un momento raro en nuestra historia: hoy por hoy, la humanidad tiene la capacidad de guardar casi casi toda su produccion artistica y cientifica (10 mil años de rompepelotismo y agitamiento) en un armario no mas grande que el mio. Pero no tan paradojicamente, no cuesta tanto almacenar sino comunicar tanta informacion. La alegoria es interesante.

Queda saber que mierda se hace con tanta informacion. Google tiene buenos dias delante suyo para facilitarnos la cosa.

PS: Gran parte de la inspiracion para escribir este texto vino de leer esta pagina del Apple Developer Network: http://developer.apple.com/macosx/tiger/64bit.html

About Software Architectures and the IEEE 1471 Standard

Adrian Kosmaczewski

2005-01-16

Looking for information on the topic of Software Architecture, I came accross the IEEE 1471 Standard, the “IEEE Recommended Practice for Architectural Description of a Software-Intensive System”. I must admit that I had never heard of it before. And I think there is a reason for this.

The home page of the standard, at http://standards.ieee.org/reading/ieee/std_public/description/se/1471-2000_desc.html does not really give good hints about what is the standard about, but a quick search in Google has given me enough information to post here a quick summary of it.

As far as I understood, the IEEE 1471 is a set of rules showing a common framework of methodologies and vocabulary useful to describe any “Software-Intensive” system. The expression “Software-Intensive” carries the idea that not every software project needs a formal description of its architecture (although I firmly believe that there is an implicit, hidden architect buried inside every software developer).

Before continuing with the topic of the IEEE 1471 standard, I would like to point out some facts that I feel important about Software Architecture.

For larger systems, it is fundamental to have a formal document that describes the architecture of the software being developed; and by “larger” I mean any software that is being developed by more than one person. Simply put, Software Architecture becomes self-evident when you develop version 2.0 of any software: if you can add the new features of the system easily, without too much trouble, integrating them with the old code base and even enhancing the old features, then your architecture simply rocks. Otherwise, it was screwed from the beginning.

From the previous paragraph you can infere that Software Architecture is tightly related to the evolution of software systems; how many times you have developed something to find out that you just cannot expand the initial solution a couple of months later after the initial

release? That is where you start wondering about Design Patterns, Software Architecture and plenty of other buzzwords.

But I think that this goes much further: Software Architecture is a way to describe a dynamic, comprehensive, secure and manageable system; I will describe each of these keywords and what are the implications.

1. By “Dynamic” I do not mean that your algorithms adapt well or that your exception handling scheme has great elegance (which is fine): when I say dynamic, I mean that you can adapt to the needs of your customer. Sooner or later, your customer will come up to you asking for new features. And you should (if your analysis shows that they are feasible) integrate them in the software with the least possible cost, both in terms of money and time. This is the kind of dynamism that Software Architectures enable.

2. By “Comprehensive”, I mean that the solution proposed by the Architecture of any system should, at its very least, contain a reasonable solution for every problem raised by the creation of the software. Software development is a pretty hard task, which not only can solve a problem, but can create many as well (deployment, security, deadlines, bugs, interoperability, change requests, etc). A Software Architecture has at least one proposed solution for any of these; fortunately, as time goes by, we learn more and more from the mistakes of the past, and this enables us to create better solutions in every aspect.

3. By “Secure”, I do **not** mean paranoia or marketing chit chat à la Microsoft; I mean conscience. Software integrators usually just have to know what are the risks and the threats to which the software used to create solutions is exposed: bugs in application and web servers, security holes in operating systems (no no, I am not thinking of any operating system in particular[1]) and issues like that. But ISVs (Independent Software Vendors) that provide innovative solutions, used by software integrators, have to take special care and actively maintain a security policy that enables prompt solutions to any possible threat. This is something that is poisoning Microsoft lately.

4. Finally, by “Manageable” I mean all the tasks that make your development team keep smiling throughout the project: coding standards, clear interfaces, whiteboards with colorful diagrams, well-written and updated documentation, a one-step build/deployment procedure, a clear vision of the product lifecycle, and free soda or coffee machines. This will enable you, as a Software Architect, to keep the project up and running until the last release, even if your team changes, even if you leave the company before the project is finished.

[1] http://www.microsoft.com/windows/

Every one of the four aspects I have defined above are good subjects for articles of their own... watch out for improvements in the future.

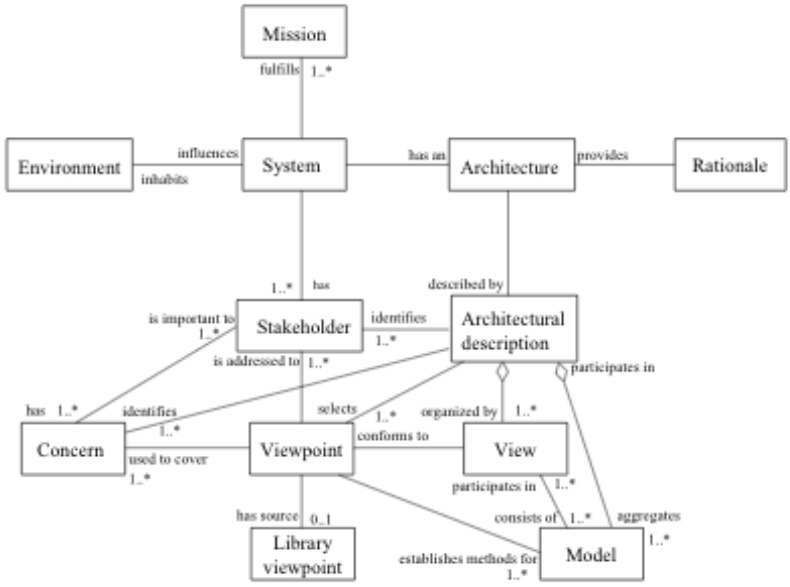
Getting back to the IEEE 1471 standard: an important fact of it is that it is a “recommended practice”: in fact, it just proposes a terminology and a set of concepts to describe an architecture, but it does not even say how to do this description in actual facts. It does not require nor define a particular description language, it does not offer any completeness criteria, and it does not force a minimal set of information in the description of the Architecture.

But it does give a formal definition for the word “Architecture”:

> “The fundamental organization of a system embodied in its components, their relationships to each other and to the environment and the principles guiding its design and evolution.”

Wikipedia[2] also has something to say about this. And Bredemeyer Consulting[3] has an impressive set of documentation.

The following graph shows the conceptual parts of the IEEE 1471 standard:



(Diagram taken from the paper at http://www.ifi.uio.no/~mmo/generic/papers/IEEE.pdf)

The central concept in the standard is the separation between the concepts of “Architecture” and “Architecture Description”; this is, I

[2] http://en.wikipedia.org/wiki/Software_architecture
[3] http://www.bredemeyer.com/

think, the most important contribution of the standard, the latter being the concrete representation of the first as a document or a set of documents.

The concepts of “Stakeholders” (not “S**k**a**t**eholder” which is actually something different) is fundamental as well: these are the roles or groups of people that might have any intervention on the creation, usage and/or management of the system, at any given level (this spans from the Developers to the Customers and the Final Users, but also embodies the Software Architect himself!). This “interest on the system” is described in the above diagram as “Concerns”, and they define a particular “Viewpoint” on the system itself.

The Architecture Description is described as a series of “Views”, which are closely related to the Stakeholders: each one of them holds a particular view on the system, and both concepts are thus closely related. For the sake of top-down decomposition of the problem domain, each View can be subdivided into several “Models”.

Finally, each Stakeholder has “Viewpoints” over a certain View: the Architecture Description identifies the Stakeholders, their concerns, and offers one and exactly one View for each Viewpoint.

Then the problem is to define this (rather fuzzy) line between the View and its Viewpoint: that is why the IEEE 1471 offers several guidelines used to define Viewpoints and their context, how to organize them and how to explain any inconsistencies among them.

For an example of a practical use of the standard in the software industry, this paper[4] (44 KB) offers an excellent insight (taken from http://www.mrtc.mdh.se/php/publ_show.php3?id=0529)

Sofware Architecture is not a new topic: Dijkstra has even talked about it in the sixties. But the formalization of its description could enable a better communication between software developers and architects, with the well-known risk that any standardization brings a reduction in creativity; I think that there is a risk, but that the benefits are interesting anyway.

[4] 0529.pdf

Who Is This Guy, Anyway?

Adrian Kosmaczewski

2005-01-17

All of a sudden you happen to be browsing some weirdo's website, plenty of blogs and some of them speak about software development.

Now, who's this Adrian Kosmaczewski, anyway? Some cheap copy of Joel[1]?

I'm glad you asked. This blog has many purposes, one of them being try to be a reference on software development and architecture at least as good as Joel's brilliant website. I will follow a similar pattern to his writings, since I've been involved in quite a few software development teams, in Argentina and Switzerland mostly, since 1997. Other valid purpose would be satisfy my urge to writing; I just love it.

First things first. Let me introduce myself; I am Adrian Kosmaczewski, from the Jastrzebiec clan[2]. Because of a series of quite[3] sad[4] facts[5] of history, my grandparents emigrated to Argentina[6], between 1929 and 1936. This explains the funny accent when I speak any language, including Spanish. People think I come from Italy, and when they see the family name, or worse yet, the Swiss passport I got 20 years ago, they just stop asking questions and shake their heads in dismay.

Lately my life has more to do with a ping-pong ball, bouncing over the Atlantic, but lately I have settled down in Lausanne, Switzerland. Who knows for how long?

In 1997 I started working as a web developer in Buenos Aires for FIS.com[7], the world's largest industrial fishing resource on the web. It was a pure dot-com, in the middle of the dot-com world. There I did a lot of ASP in VBScript, SQL Server (back then it was versions 6.5 and 7.0), JavaScript (at the time Netscape 4 was still widely used and XML was just a rumor). It's amazing to see how things have changed since. Working there was truly cool. Well, almost[8].

[1] http://www.joelonsoftware.com/
[2] http://www.polishroots.org/herbarz/jastrzebiec.htm
[3] http://en.wikipedia.org/wiki/Black_Thursday
[4] http://en.wikipedia.org/wiki/World_War_1
[5] http://en.wikipedia.org/wiki/Adolf_Hitler
[6] http://en.wikipedia.org/wiki/Chimichurri
[7] http://www.fis.com
[8] compilacion.txt

In 1999, I worked for a month as a web GUI developer for t/subcero[9]; that was fun and I could do heavy Internet Explorer DHTML... A real startup, with cool offices in downtown Buenos Aires, venture capitalists and the will to jump as far as possible into the high tech stock market.

Then in 2001 I left FIS and returned to Switzerland; the situation in Argentina was pretty tough[10]. And my job at FIS was over; the website was done, and I felt that there was no more room for improvement. Besides, the development world was indeed changing a lot, and I badly wanted to be part of the change: .NET had just appeared and people started to talk about it; XML was the next big thing; Apple had (finally!) launched Mac OS X. Looking back, I feel that I was right in taking a step back and returning to Switzerland.

In 2002 I started working as a full-time developer of the ill-fated SoftPlumbers[11], a Geneva-based startup company that tried grosso modo to reproduce Microsoft's Systems Management Server[12] on a web server. It didn't work, for many reasons, the first being management, the second being marketing, the third being fundamental architectural problems on the product. In SoftPlumbers I had my first experience with C#, .NET and software architecture: I had the great opportunity to work with my boss Adam in the creation of a whole framework to be used by web developers in the company to access the core of the system. The project was called "DataServices", and had it all I could dream of: AOP (Aspect Oriented Programming), COM Interop, WMI (Windows Management Instrumentation) integration, a Service-Oriented architecture, lightweight XML communication protocols, you name it. But it was cut off, because the company went bankrupt less than a year after I got in.

And so I found myself in the street, looking for a job again. In the meanwhile I did some training, which is something that I like to do; and then, finally, I found my current job, at Thales Information Systems[13], in Geneva.

In all these years, I've seen at least 4 different ways to manage a software team; I've worked in 4 different languages (Spanish, English, Italian and French); I've shared the joys of the dot-com boom and also the fears of what would happen after March 2000; I've had a swimming pool beside my desk and shared a big noisy room with plenty of french developers.

I've got something to say about this; if you don't mind, I will pour some thoughts in this blog, and I would be pleased that you come every so often, just to take a look, and why not, argue with me :)

[9] http://www.tsubcero.com.ar/

[10] http://archives.cnn.com/2001/WORLD/americas/12/19/argentina.riots/

[11] http://softplumbers.com/

[12] http://www.microsoft.com/smserver/

[13] http://www.thales-is.ch

And Joel[14], if you happen to read this, well, what can I say: you are simply the best! I hope you will enjoy these lines as much as I enjoy writing them.

[14] http://www.joelonsoftware.com/

Nieve

Adrian Kosmaczewski

2005-01-24

Acabo de mandarle este mensaje a una amiga, cuyo novio resbalo en la nieve y se jodio la muñeca:

“no te digo, la nieve jode todo, es un desastre, mira que la gente de este pais es idiota al ponerse contenta con esta mierda blanca que nos cagan cada invierno; no se quien fue el troglodita con el cerebro embadurnado de soretes que se le pueda ocurrir sonreir al ver semejante semen narcotizante caer desde esas putas nubes grises que cubren la unica fuente de alegria genuina que exista en este ignoto y olvidado rincon de la galaxia. he dicho.”

Y no me vengan con eso de “aaaahhh que lindo ver nieve!!!”; es HORRIBLE.

Cours Sur Architecture De Software

Adrian Kosmaczewski

2005-02-09

Hier j'ai eu l'opportunité de donner un cours sur Architecture de Software à mes collègues de Thales. Pour ceux qui le souhaiteraient, j'ai joint à ce message le fichier PowerPoint[1] que j'ai preparé pour l'occasion.

N'hésitez pas a me faire part de vos remarques et commentaires!

[1] IntroArchi.zip

El Primer Kosmaczewski? Y Los Ultimos?

Adrian Kosmaczewski

2005-02-10

Hace un tiempo encontre esta pagina donde me parece que esta la referencia a la primer persona con el apellido Kosmaczewski.

Tambien encontre que los Kosmaczewski originariamente formabamos parte del clan Jastrzębiec. El clan se establecio alla por el 900 y pico.

Y somos varios los descendientes.

The Exception to the Rule

Adrian Kosmaczewski

2005-02-13

The rainy, later snowy, Swiss weather of this weekend has made me just stay at home, and take the chance to try out a few things: that's how I made my first steps in Ruby and WebObjects. These two deserve an article of their own.

However, surfing on CNN I found an article about Ariane 5[1], which successfully took off this weekend. That made me remind of the 1996 accident, in which one of the first (if not the first, can't remember) Ariane 5 rockets just exploded 40 seconds after takeoff[2].

I just did not know that the reason of this was a software error... a bug!

A quick Google on "Ariane" brought up this interesting document[3]. In this page one can find the explanation of the software error that caused the explosion[4] and subsequent loss of (hang on) half a billion dollars (uninsured); that is, 500 million bucks! And the cause of it is (prepare your compilers) an unhanded exception due to a floating-point conversion error!

> The exception was due to a floating-point error: a conversion from a 64-bit integer to a 16-bit signed integer, which should only have been applied to a number less than 2^15, was erroneously applied to a greater number, representing the "horizontal bias" of the flight. There was no explicit exception handler to catch the exception, so it followed the usual fate of uncaught exceptions and crashed the entire software, hence the on-board computers, hence the mission.

Incredible, huh? It seems to be the most expensive software bug ever, hopefully not causing any human casualties. The explanation that follows in that article is of greater interest and I highly recommend it.

This took me to the Eiffel language; I had heard about it but didn't know any of its characteristics, nor seen any sample code. Now it seems that Eiffel introduces a language-level syntax that allows what

[1] http://www.cnn.com/2005/TECH/space/02/12/ariane.launch.ap/index.html
[2] https://www.youtube.com/watch?v=PK_yguLapgA
[3] http://archive.eiffel.com/doc/manuals/technology/contract/ariane/page.html
[4] https://www.youtube.com/watch?v=PK_yguLapgA

the Eiffel guys name “Design by Contract (TM)” (watch your step, it’s a trademark). In spite of the marketing blah blah that goes around these fancy words, I was caught by the definition and the syntax of Eiffel, that takes the definition of a method to another level. Take a look at this:

```
method_name (parameter_name: INTEGER): INTEGER is
require
    parameter_name <= some_maximum_value
    -- more conditions, if needed...
do
    -- code of the method here
ensure
    -- postconditions that must always be met
    -- no matter what happens, here
end
```

As shown in the previous example, an Eiffel method can define “require” and “ensure” blocks that contain pre- and post-conditions that must be met by the method before any processing of the “do” block. Really neat. What happens if these conditions are not met? Well, you’ve got your exception popping off the stack. Meet the conditions, and your method will exit cleanly and silently.

That’s your “contract”: take out the “do” block from your method, and you will see a couple of lines that describe a handshake contract between your method and any client that uses it. Even non-engineers can see it and understand what’s going on, without having to deal with “how” things are implemented.

And how does this relate to our day-to-day activities? Of course neither me nor my colleagues work on Eiffel (yet). But, we do create software, we do handle exceptions (do we?), and we do lose money, credibility and faith every time there’s an unhanded exception in our software. These exceptions cost us a lot: that “Design by Contract (TM)” thing can positively help us, just by rethinking the way we build our software.

Here’s my idea: even if we don’t use Eiffel, our good-old Algol-related languages can be used in pretty much the same way as Eiffel behaves, but of course it requires some of our own brainy CPU time. The trade off is simply a much clear interface that fits into a higher level, architectural view of the system, explicitly stating the valid ranges of execution for our code, and helping out in setting unit and integration tests.

Consider what an “exception” is: it is simply not the rule. For example, if a method must read the contents of a file in a remote network location, several things can happen:

- The network connection may not open;
- The network connection may close unexpectedly in the middle of the transfer because of some proxy between our code and the

file server;
- The file might not be there at all;
- The file might be there but might not be read because of security or sharing reasons;
- Etc…

Seen like this, it seems like a miracle that our routine works at all, right? Well, even with all those problems, our method should take a string (the filename) and return a stream of bytes (the binary contents of the file) no matter what happens in the middle. The rest, simply has nothing to do with the "normal" workflow of the application: those situations are handled as exceptions. Excuse me, I will repeat and slightly change the last sentence to make the point: those situations must be handled as exceptions.

This way, we have defined the contract for our method: it should expect some kind of string (thus a simple regular expression could help us tell if the string is a valid filename -without forbidden characters- or not) and should return some byte stream (hopefully encoded in a way that our hardware and software can read it!).

Our code, the interesting part of it, can then trust its environment and perform its operations in the safest possible way.

Eiffel's capabilities not only deal with methods but also with classes as a whole: it allows class developers to define "invariants", that is, conditions (such as boundaries) that class fields should respect at any time during the lifetime of an instance. These "invariants" can also be inherited by the subclasses, thus providing not only behavior and structure inheritance but also validation rules inheritance. This is an extremely powerful concept, that I wish C# had built-in from the beginning (in fact I would trade .NET generics for some time of Eiffel-like capabilities…!).

To achieve similar results in .NET, a couple of years ago I was involved in the development of an execution runtime that we called "DataServices". It allowed to decorate method signatures with .NET Attributes, having a special infrastructure perform pre- and post- processing on the method input and output. The underlying idea in Eiffel and DataServices was the same, and it's really close to the ultimate goal of AOP (Aspect Oriented Programming): provide processing infrastructures that allow to create bigger, reusable class frameworks.

Using DataServices, you could define methods like this:

```
[Log()]
[RequiresRole(Role.Admin)]
[Database(ConnectionType.SQLServer)]
public bool CreateRecord([RegExp("[a-z]*")] string name, [Range(1, 99)] int age
{
    // just open the connection and insert, no further checking needed!
}
```

As you can see the CreateRecord method contains .NET attributes that

define the valid ranges of execution for the parameters, and has some other attributes that define pre- and post-conditions to be checked prior to execution. This allowed us to centrally manage a quite large framework (~20 classes, ~100 quite complex methods) and centralize the debugging, security, logging and parameter checking routines into a single location.

I think that we achieved a similar goal that the one promoted by Eiffel's designers, while theirs is much, much more elegant :)

I can only recommend watching the following presentations on the Eiffel website, which will give you a better overview of what's Eiffel about: http://www.eiffel.com/developers/presentations/dbc/partone/player.html?slide= http://www.eiffel.com/developers/presentations/dbc/parttwo/player.html?slide=

By the way, I have also asked for my trial copy of EiffelStudio, I got curious about it :)

Credits: the QuickTime video linked in this article[5] comes from a CNN.com article dating from 1997[6].

[5] https://www.youtube.com/watch?v=PK_yguLapgA
[6] http://www.cnn.com/TECH/9710/30/ariane.launch/

Thin as a WEBrick

Adrian Kosmaczewski

2005-02-16

Estuve probando un nuevo lenguaje de programacion que no conocia: Ruby. Interesante pero medio criptico a veces; es un derivado de Perl (de ahi viene el nombre, de “perla” a “rubí”) y de la familia del Algol (como todos los lenguajes “clasicos”: C, C++, Java, PHP, etc).

Caracteristicas principales:

- Es un lenguaje de script interpretado, no compilado;
- Orientado totalmente a objetos; todo, incluso los numeros, son objetos;
- las instrucciones no se separan con punto y coma sino con nuevas lineas (como el BASIC);
- No tiene tipos (como JavaScript);
- No se necesita declarar las variables (identico a PHP pero diferente de JavaScript);
- Las variables no necesitan tener un “$” delante (como sucede en Perl y PHP);
- Tiene una sintaxis mas compleja de lo comun para los bucles;
- Tiene soporte nativo de expresiones regulares, obviamente (como JavaScript y Perl);
- Se pueden definir clases, con una sintaxis simple y “clasica” (keyword “class”);
- Para definir funciones, se usa la palabra clave “def”; tambien para definir metodos dentro de las clases (como en Lisp, y con la misma posibilidad de definir funciones dentro de funciones, identico a Lisp y a JavaScript);
- Tiene una biblioteca de funciones de la hostia;
- La distribucion oficial pesa solamente (atencion!) 2 MB, nada mas, con todo incluido, es todo el codigo fuente!;
- Es open source y gratarola;
- Existe para cualquier sistema operativo, incluso Windows y Mac OS X, obviamente.

Se parece a esto:

```
# esta  linea con numeral es un comentario
# aca declaro y defino una variable string
str = "abc"
puts str.length            # =>  3
# lo mismo para un array
```

```ruby
arr = [10, 20, 30, 40, 50]
puts  arr.length           #  => 5
```

“puts” es la instruccion para “escribir” algo en la pantalla; “gets” es para leer. Nada demasiado complicado por ahora.

Tambien se pueden hacer cosas bastante complicadas, con dos lineas de codigo, usando una potente libreria que nada tiene que envidiarle a la de .NET, Java o Cocoa: por ejemplo abrir una conexion de red y leer informacion de un servidor remoto:

```ruby
require 'socket'
puts TCPSocket.open("server_name",  "daytime").read
```

Ruby fue creado por un japones que ya se volvio leyenda; esto fue hace unos 10 años mas o menos. Hay muchas librerias Externas disponibles en la web que se instalan facilisimo.

Para que sirve? Es un lenguaje de script, asi que se usa mucho para programacion web (en servidores mayormente); ultimamente se esta usando mucho como lenguaje de enseñanza, tambien para automatizar pruebas y tests.

En el Mac OS X viene instalada la version 1.6.8, pero la ultima version es la 1.8.2; haciendo unos tests de una revista me di cuenta que necesitaba la version 1.8.2, la baje (2 MB!), la compile y la instale y anduvo al toque. Tremendo.

En la revista Dr. Dobbs del mes pasado (enero 2005) aparece el siguiente script, que permite:

1. Crear un web server en el puerto 2000 en la maquina en la que funciona el script;
2. Cargar un “servlet” como los de J2EE, para que responda a los pedidos en la pagina “blog” devolviendo el contenido de todos los archivos HTML del sitio en cuestion;

He aqui el codigo, reproducido sin permiso ni nada, pero bueno, esperemos que nadie se chive, de todas maneras no gano un mango con esto:

```ruby
# este script funciona con ruby 1.8.2; aparece en la edicion de
# enero del 2005 de la revista "Dr. Dobb's Journal"
# esta linea indica que estamos importando la libreria "webrick"
require 'webrick'

# aca se define una clase que representa un articulo en un website;
# cada "articulo" existe fisicamente como un archivo HTML, por eso
# el "constructor" (el metodo "initialize") toma como parametro el
# nombre del archivo.
class Article

    # estas son las variables internas (campos) de la clase
    attr_reader :file_name, :title, :body
```

```ruby
    # este es el metodo "constructor" de la clase,
    # que es llamado cada vez que se hace por ejemplo
    # "Article.new("index.html")"
    def initialize(file_name)
        body = File.read(file_name)
        title = file_name

        # aca se usan regexps para leer el archivo HTML...
        if body =~ %r{<title.*?>(.*?)</title}m
                || body =~ %r{<h1.*?>(.*?)</h1}m
            title = $1
        end
        body.sub!(%r{<body.*?>(.*)</body.*}m) { $1 }

        # le damos valores a los campos de la clase
        @file_name, @title, @body = file_name, title, body
    end
end

# definicion de un metodo de la clase "Article"
# grosso modo, este metodo hace lo siguiente:
# 1) saca la lista de archivos HTML que estan en el folder que se pasa en param
# 2) los ordena segun la fecha de creacion
# 3) para cada archivo encontrado, crea una nueva instancia de la clase "Articl
# la sintaxis para los bucles en ruby es alucinante!
def Article.list(dir)
    Dir.chdir(dir) do
        file_list = Dir.glob("**/*.html")
        sorted_list = file_list.sort_by {|name| File.stat(name).mtime }
        sorted_list.reverse[0, 10].map do |file_name|
            Article.new(file_name)
        end
    end
end

# el "<" indica una relacion de herencia con otra clase llamada
# WEBrick::HTTPServlet::AbstractServlet
class BlogServlet < WEBrick::HTTPServlet::AbstractServlet

    HEAD = "<head><title>Sample Blog</title></head>"

    def do_GET(request, response)
        articles = Article.list(@server.config[:DocumentRoot])
        content = articles.map{|a| a.body}.join("<" + "hr />")
        response.body = "<html>#{HEAD}<body>#{content}</body></html>"
    end
end

# primero creamos un webserver en el puerto 2000
server = WEBrick::HTTPServer.new(:Port => 2000, :DocumentRoot => "html")
```

```
# si el usuario tipea "CONTROL+C" el server tiene que cerrarse
trap("INT") { server.shutdown }

# aca cargamos el servlet en la memoria y arrancamos el servidor
server.mount("/blog", BlogServlet)
server.start
```

Todo esto se logra gracias a la libreria WEBrick que se baja de http://www.webrick.org/, y que, contrariamente al disco de Jethro Tull, no es “Thick as a Brick” sino mas bien “thin” ya que pesa solo 70 KB. Y el circulo se cierra, ya que estuve aprendiendo Ruby mientras escuchaba ese disco, justamente... y se explica el titulo pelotudo de este post que ya se hizo demasiado largo, como es costumbre. :)

Muy piola esto de Ruby, me gusto. Voy a seguir investigando...

Unix Expo, Remarks by Bill Gates - October 9, 1996

Adrian Kosmaczewski

2005-03-05

Taken from http://www.microsoft.com/billgates/speeches/industry&tech/uexpo.asp[1]

(…)

If we go way back in time, **Microsoft was actually the first one to go to AT&T and beg to get a nice high-volume commercial license for Unix. And for many, many years we were the highest volume licensee**, not only for our own Xenix products, but Siemens with theirs, Santa Cruz with theirs, and dozens and dozens of sub-licensees.

I have to admit, it was fairly difficult to work with AT&T back then. They simply didn't understand what they had. They didn't understand how to manage the asset, either in terms of promoting it properly or in terms of making sure that there wasn't fragmentation in how different implementations were put together. And so that vacuum in leadership created a bit of a dilemma for everybody who was involved in Unix.

Well, Microsoft stepped back and looked at that situation and said that the best thing for us might be to start from scratch: build a new system, focus on having a lot of the great things about Unix, a lot of the great things about Windows, and also being a file-sharing server that would have the same kind of performance that, up until that point, had been unique to Novell's Netware.

And through Windows NT, you can see it throughout the design. In a weak sense, it is a form of Unix. There are so many of the design decisions that have been influenced by that environment. And that's no accident. I mean, we knew that Unix operability would be very important and we knew that the largest body of programmers that

[1] https://web.archive.org/web/20040227001934/http://www.microsoft.com/billgates/speeches/industry&tech/uexpo.asp

we’d want to draw on in building Windows NT applications would certainly come from the Unix base.

(…)

Continued…[2]

[2] https://web.archive.org/web/20040227001934/http://www.microsoft.com/billgates/speeches/industry&tech/uexpo.asp

Fisura en El Tupperware

Adrian Kosmaczewski

2005-03-05

Vivir en Suiza no es cosa facil. El otro dia lo hablaba con Laurita, justamente; de lo jodido que es vivir en Suiza para alguien que “cae” aca sin conocer a nadie. Te la regalo. Incluso despues de haber pasado unos 15 años en este pais (como es el caso de Laura) es sorprendente ver la cantidad de gente que se encierra en sus mundos, nunca dejandote entrar en ellos. Gente que te conoce, que “dice” que te aprecia, que tal vez llegado el caso te dirija la palabra cuando la saludes; pero que nunca, nunca soportara que la llames por telefono a las 10 de la noche. Y que probablemente se ofenda terriblemente por este ridiculo (y aparentemente anodino) atentado a la privacidad.

No es mentira; yo he perdido amigos por el simple hecho de “caer” un domingo a la tarde para visitarlos sin prevenir. Bueh’, “amigos”, no creo que el sustantivo sea el adecuado. Digamos “conocidos”.

La privacidad. El gran descubrimiento de la sociedad suiza es la privacidad, la lejania. Lo ajeno es toxico; hay que alejarse del otro. El otro puede causar daño. He aquí la frase fundadora de la sociedad suiza. Lo suizo es “privado”, esta “privado” de humanidad, de cercania, de contacto. Es frio, lejano, y falso. Terriblemente falso. Pero tambien miedoso: es, a este momento, la unica explicacion posible que encontre a esta actitud. El miedo. El suizo tiene miedo del otro.

El otro dia fui al teatro, a ver una obra que contaba la historia de los inmigrantes italianos que, en los años 40 y 50, llegaban a Suiza para construirla, para lavar su ropa, para limpiar sus calles. Los italianos, que salian de un pais en guerra, apenas atravesaban la frontera debian desnudarse y ducharse, incluso en pleno invierno, (!) y los rociaban con desinfectante, antes de pasar los controles aduaneros. Esto de desnudarse era muchas veces traumatico para campesinos de fuertes creencias catolicas, y las fotos de la epoca recuerdan algo parecido a Auschwitz. Muchos fueron reenviados a sus lugares de origen por el simple hecho de negarse a desvestirse. Luego los explotaron, para que negarlo, dandoles los trabajos mas degradantes, pagandoles los salarios mas bajos, y obligandolos a vivir en las condiciones mas infrahumanas. Una de las frases mas significativas refleja el estado de espiritu: “ni siquiera nos dejan vivir en los inmuebles que construimos”. Los italianos la pasaron mal en Suiza, pero para muchos la opcion era morir de hambre o morir de tristeza. El ser

humano elige, generalmente, la segunda opcion, muchas veces pensando ingenuamente que sus hijos podran morir contentos y con la panza llena. Y la historia se repite.

Tal vez alguien que conozca el pais este saltando en su silla diciendo “no! La parte francesa de Suiza es mucho mas amena, humana, tal y como la parte italiana! El problema son los Suizos de habla alemana”. Y a eso, querido lector, le digo: pamplinas. Los suizos son suizos, de donde sea que vengan. Conozco gente que a los dos meses de estar viviendo en Suiza ya adopta las actitudes locales, incluso viniendo de lugares lejanos como China o Zimbabwe, y esto en Ginebra, Zürich o Lugano.

Alguien que conozco debe estar sonriendo mientras lee estas lineas. Si, Suiza es contagiosa. Uno puede facilmente caer en estos modos despegados, lejanos, altaneros. Es ciertamente facil, tentador; la irresponsabilidad perfecta, el anti-contacto, el juicio en la mirada. El gesto lento, majestuoso, la actitud pajera y gatopardista. La puntualidad inutil, las sonrisas forzadas, los peinados ridiculos, y un sentido de la estetica absolutamente repugnante.

Para usted, querido lector, que se piensa que uno es un winner nomas porque le toco vivir en Suiza, creale a quienes estan adentro: esto no es facil. Conozco muchos que llegaron aca, se quedaron un tiempo y despues se rajaron a la mierda. Lo se bien; yo tambien lo hice alguna vez.

Pero Suiza se esta fisurando. El “tupperware” esta viejo, y ya no protege de la intemperie. La Matrix tiene muchas fallas. Llamese como quiera, pero esta situacion es totalmente insostenible, incluso para los Suizos. Obviamente, el establishment aca tiene una fuerza que no tiene en toda la galaxia; aquellos que tienen buena vida en Suiza (y los hay) tienen tal vez el mejor nivel de vida del mundo occidental, lo cual es mucho decir; mucho mas que en Suecia, mucho mas que en Canada o Australia (no digo Estados Unidos, ya que es sabido, el nivel de vida alla fue siempre de una franca decadencia desde el primer dia).

El cambio viene de abajo; como siempre. No de arriba: de abajo.

Primavera

Adrian Kosmaczewski

2005-03-21

Historias largas, pesadas, combinandose y dejandose llevar sin mas sentido. Dias identicos, grises, frios. El calor artificial de una frazada, o de un radiador, y la diaria gilada sin sentido.

Arboles pelados, sin hoja ni capullo; aire sin abejorros ni pichones, sin aromas y sin alegria. Ausencias. Miradas frias, gente lejana e histerica, calles deserticas dignas de Oesterheld. Nieve en las calles, demasiada nieve. Hasta el hartazgo.

Y nada en el horizonte, ni sol ni cambios. Vivir en un lugar estatico enfria los sentidos, vacia el alma de sentimientos, te deja sin ganas de seguir. El esfuerzo necesario para el aguante es enorme, y a medida que pasa el tiempo se vuelve mas dificil de mantener la frente alta. El frio quema los ojos.

Entonces, un rayo de sol. La temperatura sube, el aire se puebla de los primeros aromas de la vida y la compañia. Las cosas de la vida diaria toman un matiz insospechadamente interesante, simple, sincero. Los perfumes se sienten fuerte cuando se despiertan los sentidos. La piel se suaviza, el sol vuelve a quemar los ojos, y uno no quisiera ser tan poco cortes como para dejar pasar el momento sin acompañar el movimiento.

Si hay despertar, que haya para todos. Incluso para uno, que se refugio dentro de una carcaza durante un par de años.

Y el despertar incluye esa caricia negada (hacia uno y hacia los demas) durante un tiempo demasiado largo. Y esa caricia te remite a recuerdos lejanos, algunos lindos y otros menos, donde tus sueños se transforman en recuerdos de lo que no fue. Donde lo vivido se transformo en escudo para evitar nuevas estafas, y donde la tristeza se transformo en parte de la naturaleza diaria.

Hay fuerzas mas potentes de lo que uno cree.

Fue un largo invierno, tanto adentro como afuera de mi corazon.

MDA, Thales, Nestle, and Microsoft

Adrian Kosmaczewski

2005-03-27

I haven't blogged for a while; I must admit that this has been a busy month. In fact until June I will be working inside one of the biggest companies on Earth: Nestle. My task, as a Software Architect, is to set up the overall architecture of a worldwide, distributed application, whose details I shall not lay down here (for I have signed a non-disclosure agreement). But I can say that this is, by far, the most exciting time of my professional life; this architecture is already the biggest challenge I have ever faced.

The application, following Nestle's alignment with Microsoft technologies, will be built using the .NET platform. It will integrate inside Nestle's Enterprise Service Bus, and thus will be using special SOAP services designed by Nestle's staff to handle several cross-cutting concerns such as security, reporting,

On the other hand, my employer, THALES, has become one of the members of the board of directors of the Object Management Group (OMG). This means that a global policy of THALES from now on will be to support one of the latest and biggest standards proposed from the OMG, that is, MDA, or Model-Driven Architecture. I will be blog more on MDA soon, since next April I will attend a special MDA workshop in THALES' headquarters in Paris.

I will not go deeper in the details of MDA, which you can find here, but I will nevertheless comment my particular situation right now. The problem is the following:

- Nestle has a strong commitment to using Microsoft based technologies; this in itself is fine, since I am mostly proficient with Microsoft's tools and technologies.
- Microsoft has a slightly different way of seeing application architecture and modelling, and it has definitely nothing to do with MDA.

I hope you see my point; maybe this article will help you see it easier:

> Is that Model Driven Architecture (MDA)? No, not quite. Like MDA, we are interested in models. However, we are much less concerned with portability and platform independence than MDA, and much more concerned with productivity. While stereotypes and tags can be used to decorate UML

> sublanguages, experience suggests that much more precise language features are required to support compilation, debugging, testing and other software development tasks. Unlike MDA, we do not propose to use UML for this purpose.
>
> (…)
>
> What about interoperability? Isn't MDA right in calling it a key piece of the story? Indeed, interoperability is a key piece of the story. In our view, however, it is much more important to standardize the protocols used to specify and assemble components, than the languages used to implement them. We think the web service protocols standardized by the World Wide Web Consortium (W3C) and the Web Services Interoperability Organization (WSI) hold much more promise for enabling interoperability than UML.
>
> (…)

As you can see, Microsoft's view of things is quite different from the OMG's. Microsoft sees (and I recommend you a thorough reading of the above article) Software Factories and SOAP as the way to build the next generation Enterprise Service Bus, while MDA stresses modelling and portability; the key difference between both approaches has to do with the keywords "platform independence". Microsoft is not at all interested in that (of course not) and would rather bet on delivering productivity tools that help achieve faster time-to-market scores (which I believe is great). As soon as you stick to using Windows servers and Microsoft tools, you can get your apps up and running much faster than if you think "models".

This has raised a couple of questions in my head; these questions will be the main axis of my future work as an architect:

- What are the tradeoffs (if any) of both approaches? Or better still: how can I make them collaborate? How can I design this particular application using MDA? Is it worth it?
- How can I leverage THALES' commitment to MDA, and use this knowledge for building platform-specific solutions such as those needed by Microsoft-based organizations such as Nestle?
- Will the .NET platform grow as a standard such as J2EE? In this sense, will the CLI ECMA/ISO standardization process help? Will Mono provide an Enterprise-class platform, enabling portability and taking .NET development to a next big step towards other operating systems?
- Will Microsoft allow this to happen? I would not be that fast to say "NO"; Microsoft is going through deep internal power struggles, and it would not surprise me that, in the near future, the .NET and the Mono platforms would be backed together as a whole standard infrastructure platform (which I think will ultimately happen, since no closed systems can survive in today's market conditions). The J2EE approach (an open standard with several vendors providing implementations) is definitely interesting.

Waiting for the next big thing to happen, I will try to give answers to these questions in the near future.

About code and eggs - excuse me?

Adrian Kosmaczewski

2005-04-20

The purpose of this article is to show that the current trends in software development owe a lot to ancient mindsets, and that some good old Object-Oriented Programming (OOP) programming constructs are no longer accepted in modern business development scenarios. It serves also as an introduction to Service-Oriented Architectures (SOA), putting it into context of what has been done in the past, and what can be done in the future with it.

Procedural vs. OOP

Since the OOP programming paradigm appeared, tons and tons of books and articles have been written about the advantages of it, against "older" ways used to structure the code of your application.

Let's face it: traditional compilers (not those targeting virtual machines such as Java or .NET) produce binaries, executables targeting some platform, no matter what programming language you choose, be it procedural or object-oriented. At the end, this binary code does not care whether it was pumped up using OOP or procedural techniques; the resulting binary code may be different (and usually is) but the processor does not care about it: it's just plain vanilla executable binary code.

By the end of the seventies almost all serious system programming was done in C; this language is sometimes referred to as "portable assembly" and is available in every single development platform on Earth. It has become a de facto standard language, used to create anything from operating systems, word processors, spreadsheets, you name it. It is the language that others have copied, both in terms of look and feel and capabilities (every "curly bracketed" language over there has borrowed something from C, be it C++, Objective-C, Java, JavaScript, C# or PHP). Kernighan and Ritchie[1] can be proud.

C is a procedural language like classic ALGOL-family languages (Pascal, Basic, FORTRAN, etc), which means that you concentrate primarily in "how" you do things, rather than "what" is being done (or does it). In C you basically write kitchen recipes: you specify the ingredients, you assume the existence of at least one cook, you tell him for

[1] http://cm.bell-labs.com/cm/cs/cbook/

how long she or he should heat those ingredients, and voilà, you've got your omelette.

This is what programming was, until the seventies.

Alan Kay[2] of the Xerox PARC[3] changed everything with Smalltalk and the OOP paradigm; in any typical OOP language you no longer focus in the recipe itself, but you start "modelling your problem domain". This last phrase means it all: you start describing the classes of objects that interact (the cook, the ingredients, the pans, the resulting omelette) and then you describe the interactions among them (pan holds omelette, omelette contains eggs, cook breaks eggs, etc). Instead of focusing in the recipe, you focus in the theatre play that will ultimately deliver your omelette.

Let me tell you something that I learnt by experience: both approaches, OOP and procedural, are useful, none is "better" by any absolute means than the other. Each has its strengths and weaknesses, and you have to consider the context in which each applies better than the other; the important things to consider are the tradeoffs that you are ready to accept while building your solution, the size of it, and the underlying platform that you use to build it. Each of these parameters usually clearly determines which approach to use.

I hope that you get my point; I do not want to fuel the flames of another religious war, but this is what software development is all about.

20 years have passed since Stroustrup[4] created C++, and now we are in good shape to look backwards and see what the industry has generated:

- Lots (and lots and lots) of project failures;
- Incredible (but far, far fewer) success stories;
- A thousand different methodologies;
- The dot-com boom;
- Google.

And among all of these stories, best practices emerged out of good and bad experiences. It is now possible to find great books about software development, telling what you should and should not do in every situation of most development projects. One does not have the right of not knowing what happened before anymore.

Back to C

In plain C you would "break the egg" doing something like this:

```
Egg egg;
break_egg(&egg);
```

[2] http://www.smalltalk.org/alankay.html
[3] http://www.parc.xerox.com/
[4] http://www.research.att.com/~bs/homepage.html

```
Omelette *omelette = cook_omelette(&egg);
```

You define a variable that points to an instance of an Egg structure, then pass it as reference to a C function that will provide the egg-breaking logic. So far so good. Here there is a clear separation between the data and the procedure, but the existence of the Cook is supposed, inferred, but never disclosed (nor needed, at least in this case).

The first reaction, when translating this code to OOP, would be to represent the above C snippet into this C++ code:

```
Egg * egg = new Egg();
egg->breakEgg(); // "break" is a reserved word, after all
Omelette * omelette = new Omelette(egg);
omelette->cook();
```

In this case, the egg-breaking logic is located inside the Egg class. We might have even used the same C code (maybe not, but C++ is a superset of ANSI C, after all), providing that you replace every instance of the "&egg" parameter variable by a "this" pointer, and you would have your code up and running in no time. The same is valid for the Omelette class.

But OOP is more than a way to produce fancy code; OOP was one of the first mainstream industrial solutions to one of the worst problems in software development: source code maintenance. Source code is not something that, once compiled, disappears forever buried inside your source control system: source code is an evolutionary asset, that might as well become a huge liability if you do not take care of it from the very beginning in your design, and this is true in any programming language, be it procedural or not.

Thus, OOP must be used to create code that can be maintained; by different people, at different times, in different places. Source code must be flexible enough to allow modifications, fixes and refactoring, and strong enough to represent a complete problem domain.

Let's say that we're now in 1990: what if we want to model the whole kitchen? What if we want to model multithreaded cookers doing several omelettes at the same time? Say that your client, thanks to the success of "Omelette 1.0" (released in 1987 for MS-DOS) decides to grow up his team of cookers and want to model different egg-breaking methods (say, the French, the Brazilian and the Thai methods). What do you do?

You must do one of the most important things in software development: decouple functionality. Omelette 2.0 for Windows 3.1 would have a new object model (and hopefully you have read "The Mythical Man-Month" and won't suffer the second system syndrome). Of course, you lose backwards code compatibility (and you should refactor your application completely, which could quickly become a huge problem...) but as you will see, the benefits are worth the change.

Thus, this would be the equivalent, more maintainable code:

```
Cook * cook = new Cook();
Egg * egg = new Egg();
cook->breakEgg(egg);
Omelette * omelette = cook->cookOmelette(egg);
```

Is it correct? Is it better? And in which context? Is it so "illogical" to pretend that an egg could break itself? What does "logical" mean in this context, anyway? The discussions are endless, and I've seen every possible answer. Each OOP person will come up with its own experience and insight, and will try to convince you that her or his position is right and that others are wrong.

One can safely argue that the binary code produced by both approaches is essentially the same, and produces the same results during runtime (even if the first is slightly more efficient just because you don't have to create a "Cook" object in the heap to take care of the egg-breaking process).

My opinion is that the second C++ code is better, for a number of reasons:

- It is a more suitable view of the world; in reality, eggs don't usually break by themselves without external help; it also describes the cook, and this class might as well be used in other parts of the application for other tasks.
- Since it is a more "classic" view of the world, the code is more maintainable (the computer industry has always had important turnover rates, people come and go and you must assume that today's code will be maintained by tomorrow's people...)
- The Egg class becomes more a "data" class, without further complexity; the Cook class holds the knowledge of all the needed processes to create an Omelette.

Finally, remember the classic tradeoffs that appear when using OOP technologies:

Performance

- Usually OOP techniques are not suitable for high performance systems because of the payload needed for resolving polymorphic methods, or for heap allocation and deallocation; that's why even today, hardcore systems are coded in assembly or in C (operating systems, device drivers, embedded controllers, etc). For business applications, the primary concern regarding performance has to do with the scalability of the systems when deployed in large organizations, where they can potentially be used by thousands of users simultaneously. Special attention must be paid to stress-testing critical systems before deploying them in production scenarios.

Final binary size

- Java and .NET both try to overcome this trade-off using the concept of virtual machines, with large pre-installed class frameworks; in this case, the resulting binaries are small, since they reuse the logic contained in the framework; at the same time, they provide greater portability and solve memory-management issues off-the-box as well.

Bigger model complexity

- As Robert Glass[5] points out, a 25% problem complexity increase holds a 100% overall solution complexity increase. Be careful when you add a class to a model, and be sure that you need it.

New millennium, new paradigms

So your client has been very happy with version 2.0 of the Omelette software, and later you provided to him Omelette 95 (it used C++ COM components to encapsulate broken eggs), Omelette.NET (developed in managed C++, providing server-based, scalable, multithreaded & distributed egg-breaking components shared by both managed Windows and Web applications) and now it's time to prepare for the Omelette SOAP API. That's it, it's time for the Service-Oriented version of Omelette.

Hey, your product has really gone far; now you will provide broken eggs throughout the Internet. By the way, you are happy to have chosen a "real" programming language such as C++ and an encapsulation mechanism such as COM during the nineties, rather than a pure Visual Basic solution :) which might have proven rather difficult to port to .NET... but that's another story.

And moreover... you are now happy to have an evolutionary object model. Because now you will be sending SOAP-encapsulated eggs as messages over the Internet, and your service will just receive them, and break those eggs using that high performance, secret algorithm that you shall not show to your competitors. Can you imagine how to translate

```
Egg * egg = new Egg();
egg->breakEgg();
```

into a Service-Oriented architecture? How would you do now if your Egg class had the logic in it? How would it be serialized into a SOAP message? The answer is, you can't do that. SOAP messages only hold data, while the egg-breaking code is stored in your web service.

So, you need decoupling the data and the procedure. Quite the opposite of what many OOP advocates said back in the eighties, right? The egg does not break by itself; it will be broken, as in the real world.

[5] http://www.amazon.com/exec/obidos/tg/detail/-/0321117425/103-2831152-1341418?v=glance

And now we will handle it to someone over there to break it for us in a highly specialized manner.

This decoupling pattern is commonly used in all message-oriented architectures: it means the decoupling of the object that contains only data (something like a C struct) and another object that holds the logic (something like an object holding C functions). No longer are both entities contained inside the same structure, at least not from this high-level point of view.

And thus, the circle is closed: we have returned to a slightly more complex procedural system, that can be implemented (in a synchronous fashion) using the following code:

```
OmeletteService::Proxy * proxy = new OmeletteService::Proxy();
OmeletteService::Egg * egg = new OmeletteService::Egg();
proxy->BreakEgg(egg);
OmeletteService::Omelette * omelette = proxy->CookOmelette(egg);
```

At the end of the SOAP call, our egg variable contains an Egg instance with the "broken" flag set to true, and maybe some more information inside.

And the Cook? Well the Cook is inside the OmeletteService, somewhere on the network; it's like you had a central kitchen, that only expects from you to provide some basic material (the ingredients) and they will provide you with a delicious Omelette instance. The interaction between the client code and the service becomes a workflow, a set of rules that can be easily monitored and maintained, and even better: you can now offer Omelettes to any system that talks SOAP…

Conclusion

Somehow, the last code snippet (the SOAP call) is extremely similar to the C code shown above. And we are using the most advanced technologies! This is confusing and marvellous at the same time; we have gone through the whole circle of development, from pure procedural, to pure OOP, to a mix of both that enables the writing of more complex, distributed applications.

The new paradigm of the Service-Oriented Architecture holds the promise of information exchange, high level component reuse at network level, interoperability with otherwise incompatible systems, and stateless, loosely-coupled, message-based components, while using the primary decomposition of code and data that was at the core of procedural programming mindsets.

Remember:

- Decouple data and procedures;
- Think messages;
- Stress-test your solutions;
- Reduce dependencies everywhere;

- Don’t fear modelling and documenting your solution until you and your team feel comfortable with the object model;

Lots of things to keep in mind but remember: you can’t do an omelette without actually breaking some eggs!

New tools for open source developers - and others as well

Adrian Kosmaczewski

2005-05-22

The business of software is getting more and more complicated. In your path from coder to developer[1] you might find the need to see if someone out there has already coded what you need, hopefully in an open source project with the right kind of licence for you to be able to reuse it. And, at the same time, with the deep confidence that you are not breaking any copyright...

Well, there's Google[2] to look up for those snippets, but unfortunately it does not index code files in open source projects. And with all the X-forge websites out there (Tigris.org[3], SourceForge[4], Novell Forge[5], GotDotNet Workspaces[6], etc) it is difficult, if not at all impossible, to find the information needed.

But somebody has finally brought a solution to this problem: **Koders.com**[7] offers a search engine that indexes the most important open source websites and offers an impressive amount of source code in the same repository; Koders.com is definitely worth a look, and if you want to know more about it, here's an article on The Register[8] about it: Open source search engine trawls free code[9].

Koders.com even provides a quick reference[10] to the most important open source licences available... I would not be surprised if Google ends up eating Koders.com, as it happened with Blogger.com[11] before...

On the other side, when you start up using open source code, you might as well end up using code that you should not touch because of

[1] http://www.amazon.com/exec/obidos/tg/detail/-/078214327X/104-0869754-6039159?v=glance
[2] http://www.google.com
[3] http://www.tigris.org/
[4] http://sourceforge.net/
[5] http://forge.novell.com/
[6] http://www.gotdotnet.com/workspaces/
[7] http://www.koders.com
[8] http://www.theregister.com
[9] https://www.theregister.com/2005/05/20/koders_search_engine/
[10] http://www.koders.com/info.aspx?c=LicenseInfo
[11] http://www.blogger.com/

legal reasons. Let's face it, it is not possible to know all the possible patents that you might be violating while copy-pasting code, and if you plan to sell your product, you should take the time to parse your code with **Palamida**[12]. More information about Palamida here: Test your own software code for infringement[13].

Finally, it is worth noticing that more and more companies are offering open source code written by their internal development staff, so these two tools might become very important in the near future:

- Apple[14]
- Google[15]
- Hewlett-Packard[16]
- IBM[17]
- Novell[18]
- Sun[19]

And Microsoft... well, they have something called "Shared Source"[20], which is, in a very broad sense, another kind of open source licencing mode. Well, not really; some products (like Windows 2000 source code) are offered only under very special conditions, but for example, the Shared Source CLI[21] (Common Language Infrastructure, aka .NET for Unix) is offered for download without much more than registering using your Passport account.

[12] http://www.palamida.com/

[13] https://www.theregister.com/2005/05/15/test_your_code_for_software_infringement/

[14] http://www.apple.com/opensource/

[15] http://code.google.com

[16] http://opensource.compaq.com/

[17] http://www-136.ibm.com/developerworks/opensource/

[18] http://www.novell.com/offices/opensourcecenter.html

[19] http://www.sunsource.net/

[20] http://www.microsoft.com/resources/sharedsource/default.mspx

[21] http://msdn.microsoft.com/net/sscli/

Semana

Adrian Kosmaczewski

2005-06-20

Parece que los martes, miercoles y jueves no son interesantes musicalmente. La papa esta del viernes al lunes. La vida transcurre mas intensa en esos dias.

Los viernes, todo empieza a las 3 AM con Seru Giran, triste, muy triste, como Siglotreinta. Pero la cosa despues se arregla, despues de todo los de The Cure se enamoran siempre un viernes.

El dia mas polenta es el sabado. Todo empieza con la melancolia de The Specials, y asi llegamos al sabado a la noche, en que Elton John dice que es bueno pelearse, pero no a las 10 y cuarto, cuando reaparece The Cure. El mas lindo es el sabado a la noche de Francis Cabrel, en Frances, y el relato de un amor, efimero como la llama de una vela; Phil Collins nos lleva de la mano a la madrugada del domingo.

El domingo puede ser sangrante como el de U2 o puede ser simplemente un domingo como el de Bowie. Una preparacion, un contemplar el fin del descanso.

El lunes a la madrugada es de los Abuelos de la Nada, sonriendo complice de amor, o un Chelsea Monday de marillion, melancolico y triste.

El resto de la semana, no se.

Feliz semana.

State of the Art

Adrian Kosmaczewski

2005-07-06

This article is a copy of a research work I did today, to draw a map of today's development technologies. It is not finished (it will never be, actually) but I think it is rather interesting. Hope you find it interesting too :)

Technology Chart

Because software development **is** an art... the following table gives a quick overview of available object-oriented application blocks & frameworks, with references. This list is not intensive and will most likely change in the future.

	Java[1]	.NET[2]
Unit Testing	JUnit[3]	NUnit[4], NMock[5], TestDriven.NET[6]
Code Documentation	Javadoc[7]	XML Code Comments[8] + NDoc[9]
Build tools	Ant[10]	NAnt[11]

[3] http://junit.sourceforge.net/
[4] http://www.nunit.org/
[5] http://www.nmock.org/
[6] http://www.testdriven.net/
[7] http://java.sun.com/products/jdk/javadoc/
[8] http://msdn2.microsoft.com/library/b2s063f7(en-us,vs.80).aspx
[9] http://ndoc.sourceforge.net/
[10] http://ant.apache.org/
[11] http://nant.sourceforge.net/

	Java[1]	.NET[2]
Persistence	Hibernate[12], iBATIS[13]	NHibernate[14], Olero ORM[15], iBATIS[16], NPersist[17]; Article about Solutions for object persistence in the .NET architecture[18]
Web Applications	Spring Framework[19], Struts[20], Velocity[21], JavaServerFaces[22]	Spring.NET[23], ASP.NET[24], Castle Project[25]
Smart Clients	JFC/Swing[26]	Windows Forms[27]
Aspect Oriented Programming	AspectJ[28]	AspectSharp[29], SetPoint[30]
AJAX	For Java[31], JSON-RPC[32]	For .NET[33]
Logging	Log4j[34]	log4net[35]

[12] http://www.hibernate.org/
[13] http://ibatis.apache.org/
[14] http://nhibernate.sourceforge.net/
[15] http://www.olero.com/OrmWeb/
[16] http://ibatis.apache.org/
[17] http://www.npersist.com
[18] http://www.techmetrix.com/trendmarkers/publi.php?P=01042
[19] http://www.springframework.org/
[20] http://struts.apache.org/
[21] http://jakarta.apache.org/velocity/
[22] https://javaserverfaces.dev.java.net/
[23] http://www.springframework.net/
[24] http://asp.net/
[25] http://www.castleproject.org/
[26] http://java.sun.com/products/jfc/index.jsp
[27] http://www.windowsforms.net/
[28] http://eclipse.org/aspectj/
[29] http://aspectsharp.sourceforge.net/
[30] http://dependex.dc.uba.ar/setpoint/
[31] http://java.sun.com/developer/technicalArticles/J2EE/AJAX/
[32] http://oss.metaparadigm.com/jsonrpc/
[33] http://ajax.schwarz-interactive.de/csharpsample/default.aspx
[34] http://logging.apache.org/log4j/docs/
[35] http://logging.apache.org/log4net/

	Java[1]	.NET[2]
IDE	Sun Java Studio Creator[36], Eclipse[37], Borland JBuilder[38], IntelliJ IDEA[39], Apple Xcode[40], NetBeans[41], JCreator[42], BlueJ[43], NetComputing AnyJ[44], DrJava[45]	Visual Studio.NET[46], Macromedia Dreamweaver[47], Eclipse[48], Borland Delphi[49], #develop[50], Mainsoft Visual MainWin for Linux[51], ASP.NET WebMatrix[52], CodeSmithStudio[53]
Business Rules Engine	Jess[54], JxBRE[55], ILOG[56]	NxBRE[57], InRule[58], ILOG[59]

	PHP[60]	JavaScript[61]
Unit Testing	PhpUnit[62]	JSUnit[63]

[36] http://www.sun.com/software/products/jscreator/index.xml
[37] http://www.eclipse.org/
[38] http://www.borland.com/us/products/jbuilder/index.html
[39] http://www.jetbrains.com/idea/
[40] http://developer.apple.com/tools/xcode/
[41] http://www.netbeans.org/
[42] http://www.jcreator.com/
[43] http://www.bluej.org/
[44] http://www.netcomputing.de/
[45] http://drjava.sourceforge.net/
[46] http://msdn.microsoft.com/vstudio/
[47] http://www.macromedia.com/software/dreamweaver/
[48] http://www.eclipse.org/
[49] http://www.borland.com/us/products/delphi/index.html
[50] http://www.icsharpcode.net/OpenSource/SD/
[51] http://dev.mainsoft.com/Default.aspx?tabid=45
[52] http://www.asp.net/webmatrix/
[53] http://www.codesmithtools.com/
[54] http://herzberg.ca.sandia.gov/jess/
[55] http://sourceforge.net/projects/jxbre/
[56] http://www.ilog.com/products/jrules/
[57] http://www.agilepartner.net/oss/nxbre/
[58] http://www.inrule.com/
[59] http://www.ilog.com/products/rulesnet/
[1] http://java.sun.com/
[2] http://msdn.microsoft.com/net/
[62] http://phpunit.sourceforge.net/
[63] http://sourceforge.net/projects/jsunit/

	PHP[60]	JavaScript[61]
Code Documentation	PHPDoctor[64]	JSDoc[65]
Build tools	n/a	n/a
Persistence	Propel[66]	n/a
Web Applications	n/a	n/a
Smart Clients	PHP-GTK[67]	XUL[68]
Aspect-Oriented Programming	aoPHP[69]	JavaScript and AOP[70]
AJAX	JPSPAN[71], Flexible AJAX[72], AjaxAC[73]	n/a
Logging	log4php[74]	n/a
IDE	Eclipse[75], NuSphere[76], IDE.PHP[77], Macromedia Dreamweaver[78]	Eclipse[79], Macromedia Dreamweaver[80]
Business Rules Engine	n/a	n/a

[64] http://phpdoctor.sourceforge.net/
[65] http://jsdoc.sourceforge.net/
[66] http://propel.phpdb.org/trac/
[67] http://gtk.php.net/
[68] http://www.mozilla.org/projects/xul/
[69] http://www.aophp.net/
[70] http://www.jroller.com/comments/deep?anchor=aop_fun_with_javascript
[71] http://jpspan.sourceforge.net/wiki/doku.php
[72] https://sourceforge.net/projects/flxajax/
[73] http://ajax.zervaas.com.au/
[74] http://www.vxr.it/log4php/
[75] http://www.eclipse.org/
[76] http://www.nusphere.com/
[77] http://www.ekenberg.se/php/ide/
[78] http://www.macromedia.com/software/dreamweaver/
[79] http://www.eclipse.org/
[80] http://www.macromedia.com/software/dreamweaver/

	Apple Macintosh[81]	Microsoft Alternative[82]
Unit Testing	OCUnit[83]	Visual Studio Team System[84]
Code Documentation	HeaderDoc[85]	n/a
Build tools	Ant[86]	MSBuild[87]
Persistence	CoreData[88], DataCrux[89]	ObjectSpaces[90]
Web Applications	WebObjects[91]	Classic ASP (unsupported & unsupportable)
Smart Clients	Cocoa[92]	Avalon + XAML[93], MFC[94]
Aspect-Oriented Programming	AspectCocoa[95]	Native .NET functionality[96]
AJAX	(idem Java)	Announced[97]
Logging	log4cocoa[98]	Enterprise Library's Logging Application Block[99]

[60] http://www.php.net/
[61] http://www.mozilla.org/js/
[83] http://www.sente.ch/software/ocunit/
[84] http://lab.msdn.microsoft.com/vs2005/teamsystem/
[85] http://developer.apple.com/darwin/projects/headerdoc/
[86] http://ant.apache.org/
[87] http://lab.msdn.microsoft.com/vs2005/teams/msbuild/default.aspx
[88] http://developer.apple.com/macosx/coredata.html
[89] http://datacrux.com/datacrux/
[90] http://msdn.microsoft.com/data/objectspaces.aspx
[91] http://developer.apple.com/webobjects/
[92] http://developer.apple.com/cocoa
[93] http://msdn.microsoft.com/Longhorn/understanding/pillars/avalon/default.aspx
[94] http://msdn.microsoft.com/visualc/
[95] http://www.ood.neu.edu/aspectcocoa/aspectcocoa.pdf
[96] http://msdn.microsoft.com/msdnmag/issues/02/03/AOP/default.aspx
[97] http://www.theserverside.net/news/thread.tss?thread_id=34897
[98] http://sourceforge.net/projects/log4cocoa/
[99] http://msdn.microsoft.com/library/en-us/dnpag2/html/entlib.asp

	Apple Macintosh[81]	Microsoft Alternative[82]
IDE	Apple Xcode[100], Metrowerks CodeWarrior[101]	n/a
Business Rules Engine	n/a	n/a

	C++[102]	Ruby on Rails[103]
Unit Testing	CppUnit[104]	Test::Unit[105]
Code Documentation	CppDoc[106]	RDoc[107]
Build tools	GNU Make[108]	Rant[109]
Persistence	Progress EdgeXtend[110]	Ruby on Rails[111]
Web Applications	ASP.NET[112], C++ Server Pages[113]	Ruby on Rails[114], Comparison of Ruby on Rails against Spring + Hibernate[115]
Smart Clients	n/a	n/a

[100] http://developer.apple.com/tools/xcode/
[101] http://www.metrowerks.com/mw/default.htm
[81] http://developer.apple.com/
[82] http://msdn.microsoft.com/practices/
[104] http://cppunit.sourceforge.net/
[105] http://www.ruby-doc.org/stdlib/libdoc/test/unit/rdoc/classes/Test/Unit.html
[106] http://www.cppdoc.com/
[107] http://rdoc.sourceforge.net/
[108] http://www.gnu.org/software/make/
[109] http://rubyforge.org/projects/make/
[110] http://www.progress.com/realtime/products/edgextend/index.ssp
[111] http://www.rubyonrails.com/
[112] http://www.asp.net
[113] http://www.micronovae.com/default_csp.html
[114] http://www.rubyonrails.com/
[115] http://www.theserverside.com/articles/article.tss?l=RailsHibernate

	C++[102]	Ruby on Rails[103]
Aspect Oriented Programming	AspectC++[116]	AspectR[117]
AJAX	n/a	Ruby on Rails[118]
Logging	log4cpp[119]	log4r[120]
IDE	Eclipse[121], Apple Xcode[122], Metrowerks CodeWarrior[123], Microsoft Visual C++[124], Borland C++ Builder[125], Bloodshed Software Dev-C++[126]	Mondrian[127], Arachno Ruby[128], Eclipse[129]
Business Rules Engine	ILOG[130]	n/a

Global information

Some information about the object-oriented application frameworks above:

[116] http://www.aspectc.org/
[117] http://aspectr.sourceforge.net/
[118] http://www.rubyonrails.com/
[119] http://log4cpp.sourceforge.net/
[120] http://log4r.sourceforge.net/
[121] http://www.eclipse.org/
[122] http://developer.apple.com/tools/xcode/
[123] http://www.metrowerks.com/mw/default.htm
[124] http://msdn.microsoft.com/visualc/
[125] http://www.borland.com/us/products/cbuilder/index.html
[126] http://www.bloodshed.net/devcpp.html
[127] http://www.mondrian-ide.com/
[128] http://www.ruby-ide.com/
[129] http://www.eclipse.org/
[130] http://www.ilog.com/products/rules/
[102] http://gcc.gnu.org/
[103] http://www.rubyonrails.com/

Platform	Main Programming Languages	Multiple	Single Inheritance	Late-binding	Open source	ISO Standard	Inspired from
Java	Java	✓	✓	✗	✗	✗	NeXT OpenStep, C++
.NET	C#, Visual Basic.NET, Java, JavaScript, C++	✗	✓	✗	✗	✓[131]	Java, COM
PHP	PHP	✓	✓	✓	✓	✗	C
JavaScript	JavaScript	✓	✓	✓	✗	✓[132]	Self, C (NOT from Java!)
Cocoa	Objective-C[133], Java, C++	✗	✓	✓	✗	✗	NeXT OpenStep, Smalltalk
C++ STL + GCC	C++	✓	✗	✗	✓	✓[134]	C
Ruby on Rails	Ruby[135]	✓	✓	✓	✓	✗	Eiffel, Ada, Perl

(Useless) Comments

As you can see, there are common patterns appearing in the table above:

- Eclipse can build whatever you want
- .NET-related project names tend to begin with the letter **N**
- PHP-related project names tend to begin with the letter **P**
- JavaScript-related project names tend to begin with **JS**
- Ruby-related project names tend to begin or end with **R**

[131] C#: ISO 23270, CLI: ISO 23271 & 23272
[132] ISO 16262
[133] http://developer.apple.com/documentation/Cocoa/Conceptual/ObjectiveC/
[134] ISO 14882
[135] http://www.ruby-lang.org/

- Logging frameworks have all the same **log4xxx** name
- Java-related project names tend to… come from Indonesia :)

From an historical point of view, however, the fact that most object-oriented frameworks (that is, those not based on a scripting language) use single inheritance is interesting, and not casual: Objective-C, used for the development of the software created for the NeXT workstation, introduced the notion of “interfaces” first (originally named “protocols”), and this idea was taken lately to Java and .NET.

Que Queres Que Te Diga

Adrian Kosmaczewski

2005-07-12

Ultimamente estoy en un periodo de creacion particularmente fuerte. Digo, nada ultimamente artistico, sino mas bien logico / matematico. Hay algo de artistico ahi adentro tambien, creo yo. Buscar arte en el software es una idea que persigo desde hace años. Y trato de mantener este estado de espiritu a toda costa. No es facil.

Siento que estoy encontrando instintivamente soluciones a problemas que no sabia como resolver desde hacia tiempo, y despues resulta que estoy leyendo un libro, y resulta que la solucion que encontre se llama “metodo pirulo”, y resulta entonces que me doy cuenta de que tengo las neuronas alineadas con cierta manera de pensar. Es lo que dije alguna vez con respecto a la programacion, me “brota”, “veo” las soluciones a los problemas. El resto se vuelve despues un tema de enseñar, mostrar, debatir, comparar.

Y es ahi donde me trabo. No porque me cueste contar lo que hago o aprendo, sino porque lo que me cuesta… es encontrar interlocutores. No digo ya “validos” o no, no pasa por ahi; pero un interlocutor que no tome mis ideas como una invasion o una amenaza politica a su status quo pedorro.

Cuando uno pone en jaque el diseño de un sistema, hay dos respuestas posibles, “clasicas”, de la persona que tengo enfrente:

- Discutir sobre el sistema en si, considerandolo como perfectible
- O pensar que se pone en jaque a la persona que es o fue responsable de el

Es terrible ver que la opcion 1 es la que ocurre menos seguido de las dos. Me paso hace un rato, hablando con mi jefe de un sistema que tenemos en el laburo para controlar los horarios de los programadores (programa que aborrezco absolutamente y que destruye lo poco de creatividad que hay en la empresa). El chabon defendiendo un sistema mal hecho… nomas porque si! Sin razon! Y obviamente tapando mis argumentos (tecnicos) con otros que nada que ver! (como ser cuestiones politicas y el consabido “y, pero es asi como hacemos desde siempre”)

Enfin, el sistema en cuestion tiene dos audiencias principales:

- (de alguna manera) los programadores, que lo usan para (ejem)

anunciar cuantas horas trabajaron durante el mes en tal o tal proyecto
- (principalmente) los contadores, para calcular cuanto costamos a la empresa y echarnos si fuese menester

El sistema en cuestion, entonces, tiene una interfaz de mierda para la poblacion 1, mientras que es utilisimo y tremendamente bien hecho para la poblacion 2. Entonces los que nos encontramos en el sector 1 (notese la persona verbal) lo usamos mal, con bronca, y los de la poblacion 2 se quejan a nuestros superiores quienes nos castigan con mas controles. Entonces, los del grupo 1, que somos los que en ultima instancia hacemos el laburo, y que para hacer el laburo necesitamos tranquilidad, libertad de espiritu, comodidad para escribir buen software, nos encontramos conque nuestros gerentes (que manejan BMW, cobran 2 veces mas que nosotros y no logran un solo contrato como la gente desde hace 4 meses) nos castigan, no porque hacemos mal nuestro laburo... sino porque no rellenamos correctamente un puto formulario web de mierda con la descripcion de los horarios!!!

Resultado:

- La calidad de lo que producimos es pesima
- Perdemos clientes, porque en ultima instancia, lo que el cliente ve a largo plazo es lo que hacemos (notese nuevamente el tiempo verbal)
- Aquellos programadores que eligen anotar bien los horarios, pero que les importa un comino lo que hacen (ni siquiera digo que tengan ganas de hacer este oficio), son bien vistos por el "management"
- 15 empleados menos desde que empieza el año, clientes que desaparecen, contratos perdidos, desgano, falta de entusiasmo, "radio pasillo", calidad... calidad? Dije calidad? Nooooo...
- La empresa tiene una reputacion de mierda, y no se mejora

Algo me hace preguntarme, cual fue el bicho que me pico para aceptar un contrato ahi??? Me siento literalmente estafado. Por favor, si saben de algun puesto de arquitecto de software .NET en la zona del lago Leman[1], avisen. Mi CV[2] siempre esta al dia ;)

Me siento como Joel cuando trabajaba en su anterior empresa[3], Juno[4], y tenia sus "battles of cojones" con el management (si tienen tiempo, leanse un par de anecdotas de este maravilloso website que es Joel on Software[5], es tremendo. Incluso tiene traducciones al español de varios articulos[6]).

[1] http://www.viajar.com/reportajes/$f=211732
[2] /resume/
[3] http://spanish.joelonsoftware.com/articles/TwoStories.html
[4] http://www.juno.com/
[5] http://www.joelonsoftware.com/
[6] http://spanish.joelonsoftware.com/

Open Letter to the People in London

Adrian Kosmaczewski

2005-07-23

what i feared most was that, because of the “terrorist” attacks, your (our) “liberties” might be cut off. you know, in this world you are free to be fired from your job, you are free to choose those that will decide for you, you are free to choose to be a slave. now, not even that. they will decide even the moment and reason of your death.

for the sake of national security. the security of who?

yesterday you saw a clear sign of this; a guy, without any “visible” link to whatsoever, was literally shot down (5 gunshots! in the head!!) because he didn’t stop? and nobody reacts?? where are the people condemning the fact of being “protected” (cough) by people who can shoot you just because they “think” (cough) you are dangerous???

do you really believe that the mi5 did not know about the attacks on july 7th? did you know that benjamin netanyahu (israeli finance minister) was told not to leave his london hotel that very morning, by the secret services? he had to give a conference not far from where one of the bombs exploded. cia knew about september 11th, the mossad knew about madrid.

or have we eaten and forgotten that stupid image of a fireman finding an arab passport near ground zero on september 11th???

for god’s sake, be careful. i thought that this kind of things only happened in my beloved argentina, but now you will be able to see the worst of horrors; terrorism of state.

i remember hearing bombs exploding in buenos aires when i was 5 years old.

please keep your mind open and learn from the spanish people, that moved to the streets asking to stop all violence, and first of all, the retirement of the spanish troops from places where they should not have been in the first place.

at least that.

there is no way that violence in london will stop any time soon if you, the londoners, don’t go to the streets and ask those in power to step out. if this is democracy (cough), this is how this should be. and not,

as i read in the papers, by “following their day-by-day duties like nothing happened”. the ostrich attitude, buring the head in the ground, is definitely not an option. i do not buy that.

what will happen if the attacks continue? will you remain in your homes waiting for the end of hostilities? that is exactly what they want, whoever they are.

this is a war, for god’s sake. you cannot stop a war with another one. it cannot be done. we are all going to die. but all together we can say no, and this is the moment to do it. and all changes begin with a “no”, loud and clear.

take really care, please. i do care about what’s going on. please ask yourselves who is your enemy.

pass this over to whoever cares.

Inversion of Control, Ruby and Rails

Adrian Kosmaczewski

2005-07-31

Next week I will be in Belgium working with the Thales team in Brussels, building a new software solution (for a customer of the public sector that I cannot disclose here) using the following technologies:

- Spring Framework for .NET[1]
- NHibernate[2]

Personally, this seems like a rather new (Microsoft-less) way of doing a .NET application, and I like this! To understand what Spring is, I dived into Martin Fowler's Inversion of Control (IoC) / Dependency Injection paper[3]... not an easy trip, believe me; but a rewarding one.

The concept of IoC seems daunting at the beginning, but I found it rather simple (after the fifth or sixth time I read the paper, I must admit). The idea behind it, simply put, is the one that distiguishes a function library from a framework: while the first simply provides a set of disconnected black boxes, encapsulating (useful) functionality, the latter embodies the idea of protocols, callback methods and hooks, that provide not only encapsulated functionalities but also behavior and enforcement of best practices.

The basic idea of an IoC container, simply put, is: **don't call me, I'll call you**. Those of us who have done a little VB event-based programming have already found this pattern, without knowing its name; when you create a VB application (VB6, VB.NET, ASP.NET) the event-based paradigm tells you: don't provide your own event loop; just write the code you want to execute upon a button click on that event handler, and we'll call it up for you when the user clicks on the button. This practice, as simple as it seems, has unlocked the graphical development community (again, this had been invented in a NeXT computer, a couple of years before VB appeared on the market... but that's another story).

Lately, this concept has been adapted to larger enterprise applications through the idea of separating the behavior of the application

[1] https://web.archive.org/web/20050729030357/http://www.springframework.net/

[2] https://web.archive.org/web/20050727030548/http://wiki.nhibernate.org/display/NH/Home

[3] https://web.archive.org/web/20050731094511/http://www.martinfowler.com/articles/injection.html

from its architecture, using configuration files specifying the classes to be loaded at runtime. This way, you can architecture your code in the best possible way (programming against interfaces, using factories, the strategy pattern and other useful design patterns); and then, later, specifying externally the classes to be loaded at runtime. This can be **extremely** useful in large applications (it usually does not make any sense in small ones), and it provides the ability to change the behavior of the system without having to touch the source code. Extremely powerful, indeed.

This is the description of the Spring.NET Framework, taken from the website[4]:

> Spring.NET is a port of the Java based Spring Framework. Spring for Java contains a lot of functionality and features, many more than Spring.NET currently offers. The initial release of Spring.NET contains a full featured Inversion of Control container. Subsequent releases will contain support for Aspect Oriented Programming (AOP), ASP.NET, Remoting, and data access. This introduction discusses each of the existing libraries in turn.

And this is the description of NHibernate, taken from the home page:

> NHibernate is a .NET based object persistence library for relational databases. NHibernate is a port of the excellent Java Hibernate[5] relational persistence tool.
>
> (If you are courious about what is Hibernate: Hibernate is a powerful, ultra-high performance object/relational persistence and query service for Java. Hibernate lets you develop persistent classes following common Java idiom - including association, inheritance, polymorphism, composition and the Java collections framework. The Hibernate Query Language, designed as a “minimal” object-oriented extension to SQL, provides an elegant bridge between the object and relational worlds. Hibernate also allows you to express queries using native SQL or Java-based Criteria and Example queries.)

The concept of IoC is also at the heart of one of the most surprising, hiped, simple and useful tools that have appeared in the developer scene this year: Ruby on Rails[6]. Based on the Ruby language, which I have already covered in another article (in Spanish)[7], this little framework encapsulates a web server, a persistence engine and an MVC application server, in a small runtime not bigger than 10 MB. **IMPRESSIVE**.

[4] https://web.archive.org/web/20050723023751/http://www.springframework.net/doc/reference/html/introduction.html

[5] https://web.archive.org/web/20050731004300/http://www.hibernate.org/

[6] https://web.archive.org/web/20050714073724/http://www.rubyonrails.org/

[7] /blog/thin-as-a-webrick/

I have gathered here some useful links about the Ruby language and Ruby on Rails, I hope that you find them useful:

Introduction

Ruby is a programming language with the following characteristics:

- Scripting language
- Full OOP (like Smalltalk)
- Dynamic binding (like Objective-C and JavaScript)
- Syntax based on Perl, Eiffel and Ada
- Exception handling
- Garbage collector
- Multithreading
- Large standard library
- Reflection capabilities
- Available for many systems:
  - Windows (95 to 2003)
  - MacOS X
  - OS/2
  - MS-DOS
  - Linux
  - BeOS
  - Many Unix flavors

The creator of the language is Yukihiro Matsumoto, in 1995. The latest stable version is 1.8.2.

Links

- Ruby - The official website[8]
- Ruby in Wikipedia[9]
- RubyForge[10]
- Ruby Central[11]
- Free book[12]
- (Another) Free book[13]

Ruby on Rails

Ruby on Rails[14] (RoR) is both an MVC-based runtime web framework, as well as a set of helper scripts. RoR helps developers build websites without having to use configuration files or an instance of Apache or

[8] https://web.archive.org/web/20050731011936/http://www.ruby-lang.org/en/

[9] https://web.archive.org/web/20050724011546/http://en.wikipedia.org/wiki/Ruby_programming_language

[10] https://web.archive.org/web/20050731024646/http://rubyforge.org/

[11] https://web.archive.org/web/20050730090512/http://www.rubycentral.com/

[12] https://web.archive.org/web/20050731003541/http://www.rubycentral.com/book/

[13] https://web.archive.org/web/20050729083204/http://poignantguide.net/ruby/

[14] https://web.archive.org/web/20050714073724/http://www.rubyonrails.org/

IIS in their machine. RoR enforces the separation of code following the MVC (Model-View-Controller)[15] design pattern.

Features

- Caching
  - per page
  - per action
  - per fragment
- Data validation
- Database transactions
- Unit testing
- API for creating new generators
- API for security
- AJAX support

Related projects

- For Java: Trails[16]
- For .NET: MonoRail
- By Microsoft: Atlas Project[17]

Sample RoR applications[18]

Rails Day is a competition which gives teams of developers 24 hours to build the best web app that they can using Ruby on Rails. 191 programmers split up into 121 groups competed during the competition contributing more than 18,000 lines of code and 3,335 subversion checkins. These are the winners:

- http://124.railsday.rufy.com/record/list_items[19]
- http://75.railsday.rufy.com/[20]
- http://65.railsday.rufy.com/[21]

Websites using RoR

- http://www.tadalist.com[22]
- http://www.basecamphq.com[23]

[15] http://en.wikipedia.org/wiki/Model_view_controller

[16] https://web.archive.org/web/20070611031829/https://trails.dev.java.net/

[17] https://web.archive.org/web/20051001021708/http://weblogs.asp.net/scottgu/archive/2005/06/28/416185.aspx

[18] http://railsday.rufy.com/

[19] https://web.archive.org/web/20050719005704/http://124.railsday.rufy.com/record/list_items

[20] https://web.archive.org/web/20050731001140/http://75.railsday.rufy.com/

[21] https://web.archive.org/web/20050714004245/http://65.railsday.rufy.com/

[22] https://web.archive.org/web/20050731234504/http://www.tadalist.com/

[23] https://web.archive.org/web/20050731100121/http://www.basecamphq.com/

- http://www.43things.com[24]
- http://www.snowdevil.ca[25] (Under redesign)
- http://www.cdbaby.com[26]

Links

- http://www.rubyonrails.org/[27]
- http://www.planetrubyonrails.org/[28]
- http://en.wikipedia.org/wiki/Ruby_on_Rails[29]
- Lead developer's weblog[30]
- Tutorial[31]
- Performance compared to J2EE[32]
- Growing up an open source ecosystem[33]

Demo (July 12th, 2005)

On July 12th, 2005, I did a live demo of RoR for Thales Geneva. The demo was based on the contents of the following articles:

- http://www.onlamp.com/pub/a/onlamp/2005/01/20/rails.html[34]
- http://www.onlamp.com/pub/a/onlamp/2005/03/03/rails.html[35]
- http://www.onlamp.com/pub/a/onlamp/2005/06/09/rails_ajax.html[36]

[24] https://web.archive.org/web/20050731075640/http://www.43things.com/

[25] https://web.archive.org/web/20050730010623/http://www.snowdevil.ca/

[26] https://web.archive.org/web/20050731091336/http://www.cdbaby.com/

[27] https://web.archive.org/web/20050714073724/http://www.rubyonrails.org/

[28] https://web.archive.org/web/20050728232546/http://www.planetrubyonrails.org/

[29] https://web.archive.org/web/20050709023716/http://en.wikipedia.org/wiki/Ruby_on_Rails

[30] https://web.archive.org/web/20050816201327/http://www.loudthinking.com/arc/cat_ruby.html

[31] https://web.archive.org/web/20050730014622/http://darkhost.mine.nu:8080/~vince/rails/tutorial.html

[32] https://web.archive.org/web/20051220000333/http://weblog.rubyonrails.com/archives/2005/04/04/justin-160-gehtland-is-back-with-numbers-to-back-it-up

[33] https://web.archive.org/web/20050721015905/http://www.loudthinking.com/arc/000484.html

[34] https://web.archive.org/web/20050731003945/http://www.onlamp.com/pub/a/onlamp/2005/01/20/rails.html

[35] https://web.archive.org/web/20050731005139/http://www.onlamp.com/pub/a/onlamp/2005/03/03/rails.html

[36] https://web.archive.org/web/20050731010648/http://www.onlamp.com/pub/a/onlamp/2005/06/09/rails_ajax.html

La Frase Del Dia

Adrian Kosmaczewski

2005-08-05

“el costo social es el resultado coyuntural de procesos heteroclíticos pseudoaleatorios que se manifiestan en grupos simbióticos protodinámicos, cuando las relaciones tetralógicas e hidropónicas entre los entes que constituyen el conjunto se vuelven aerodinámicas y quadridimensionales, desde un punto de vista kantiano, claro esta; si lo vemos desde una perspectiva jungiana, vemos aparecer isomorfismos patológicos anticonstitucionales que son tipicamente marxistas.”

nunca te iras a dormir sin aprender algo.

Land of the Forbidden Maneuver

Adrian Kosmaczewski

2005-09-15

After four years of .NET, n-tier and service-oriented architectures, object-oriented programming and design patterns, I have been assigned a small... "Classic ASP" project, for Reuters[1]. Yeah, you got it, the good'ol plain vanilla ASP, VBScript and so on.

Geez.

The good news is that my technical contact at Reuters is Adam, a good friend and ex-boss of mine (yeah you can become a good friend of your boss... after the company they've worked in together[2] has gone bankrupt[3] :))

Dusting out my (by know fairly limited and mostly forgotten) ASP knowledge, the first thing I found out is that it is increasingly difficult to find information on the web about ASP; the trick was to Google like this: "Session ASP **-.NET**"[4], that is, specifying that **I do not want** .NET-related stuff in my results list...

Once I found this trick, life was easier. The messy part came right after, when I (re-)discovered that "Classic ASP"... is the Land of the Forbidden Maneuver. What I mean is this: ASP development gets harder and longer, not because of the complexity of the business rules or something like that, but just because of (among others) these Major Commandments:

- Thou shalt not store a VBScript class object in the ASP Session[5] unless it is a VBScript Array or a Scripting.Dictionary object: "A somewhat more technical problem can arise when using VBScript classes on the ASP Web server. The ASP server is multithreaded and assigns a different thread to each page request (and hence each script engine). But VBScript class instances are apartment-threaded objects, which means that they must run on the thread that created them. Therefore, attempts to use one instance of a VBScript class on two different pages via

[1] http://www.reuters.com
[2] http://www.softplumbers.com
[3] http://zefix.admin.ch/shabpdf/current/2004/193-05102004-1.pdf
[4] http://www.google.com/search?q=Session+ASP+-.NET
[5] http://www.microsoft.com/mind/1199/classes/classes.asp

storage in Session or Application scope are doubly doomed to failure."

- Thou shalt not create a disconnected ADO Recordset using JScript[6] unless you mix VBScript and JScript in the same page (which can be tricky, see the next commandment for details)
- Thou shalt not mix freely VBScript and JScript on the same page, even if they say you can[7]; compiler problems may arise[8] and the behavior of your page becomes random…
- Thou shalt not use "Server.GetLastError()"[9] freely; it can be only called from a page that has been set-up as the 500-100 error handler in IIS (which is a problem, if, like me, you don't have access to the IIS MMC of your client's server… and your client either - too long to explain…)
- Thou shalt not inherit VBScript classes[10]; code reuse is done through server-side includes: "VBScript 5.0 is not strictly an object-oriented language like C++ or Java. There is no notion of polymorphism or inheritance in VBScript 5.0. Though VBScript classes allow you to define your own objects, you cannot define an object hierarchy where, say, a Terrier class is a subclass of Dog, which is a subclass of Mammal. VBScript classes are merely a way to group data and the operations on the data together to improve encapsulation."
- Thou shalt not use any "Return" keyword to return a value from a Function; thou must use the (awkward) way of using the function name for that (what if I want to change the name of the Function afterwards, introducing a bug?); if you forget to do this, you will not get a compilation error, not even a runtime error; just a "Nothing" as a return value, and some new white hair for free.
- Thou shalt not use parenthesis when calling a Sub with more than 2 parameters (whoever came up with this idea in Microsoft should be sent to Mars to change the batteries of Spirit and Opportunity, without a space suit)
- Thou shalt define Class properties using the "Property Get", "Property Let" and "Property Set" (?) statements; the meaning of the first is quite simple, the difference between the second and third one… is arcane.
- Thou shalt always use "Set" when filling up a variable that contains… anything that is slightly bigger than a String: ADO Recordsets and Connections, COM objects of any kind, instances of your own classes… you name it. If you don't, you will get a meaningless runtime error; not even a compile-time error, a runtime one, and God bless your soul, kid.
- …

[6] http://support.microsoft.com/default.aspx?scid=kb;en-us;Q289531
[7] http://msdn.microsoft.com/library/en-us/dnvid/html/msdn_vbnjscrpt.asp
[8] http://blogs.msdn.com/ericlippert/archive/2004/02/19/76438.aspx
[9] http://support.microsoft.com/default.aspx?scid=kb;EN-US;q299981
[10] http://www.microsoft.com/mind/1199/classes/classes.asp

Anyway, you get the idea.

A last commandment of the Land of the Forbidden Maneuver, just for our salvation:

- Thou shalt not forget to scan the open-source world to finish in time your Microsoft-related project!

Update 2005-09-25: I forgot to add this one:

- Thou shalt not use the “Multiple Recordsets” feature of the ADODB.Recordset class if you use disconnected recordsets…

Balada Para Unos Cuantos Locos

Adrian Kosmaczewski

2005-09-20

Para ir a mi casa hay un tren, que sale del centro de Lausana y que se llama “LEB” (por “Lausanne - Echallens - Bercher”, que son las tres ciudades principales que toca la linea). Suelo tomarlo porque es mas rapido que el bondi y porque es una fuente inagotable de anecdotas. El trencito este pasa cuatro veces por hora cerca de casa, dos yendo para Lausana, dos yendo para Bercher, cada 15 minutos mas o menos.

Parece que mas alla de donde me bajo yo, en la parada de “Montetan”, hay un asilo psiquiatrico. Digo esto porque cada vez que me tomo el de las 16 y 35 para ir a Lausana, hay un chabon que recorre todo el tren antes de bajarse dandole la mano a cuanta persona se le cruce. Infaltablemente, viene a saludarme, me da la mano, y el mismo dialogo se repite, invariable y totalmente previsto:

– Hola!
– Que tal? – le respondo siempre con una sonrisa.
– Nos conocemos? – obviamente mi actitud lo sorprende; la gente no sonrie mucho por estos pagos.
– Claro que si!
– De donde?
– De siempre.
– Vivis en Lausana?
– Si.
– Estas casado?
– No.
– Ah bueno, chau, suerte. – y diciendo eso, se baja en la siguiente estacion, acompañado de otros tantos locos, cada cual en su mundo, muchos de los cuales no vamos al asilo.

SLOCs and Other Statistics

Adrian Kosmaczewski

2005-09-27

The Reuters' project I've previously written about is finally coming to an end, at least version 1.0. It is now "feature complete" and tomorrow it will be deployed in their own servers. That's a nice milestone!

Then I asked myself, how many lines of code have I written during this month? How many did I write per day in average? This is an unique opportunity to know, since I have been responsible for this little project from beginning to end, from analysis to design and implementation and testing, through the installation in their own infrastructure.

Just a quick reminder of the technical characteristics of the project:

- Intranet "Classic ASP" application, consisting of: 3 different screens, one big DHTML graphical editor (complex enough) with "WYSIWYG" capabilities, and a couple of minor pop-up windows. This application will be part of a larger project under the responsibility of other Reuters' teams, and provides a unique "viewing" capability for the managerial staff.
- Programming languages: VBScript, JavaScript, Transact-SQL, CSS
- Supported browsers: Internet Explorer 5.5 & 6, Mozilla Firefox (Mac and Windows) & Apple Safari (haven't tested in other browsers)
- SQL Server 2000 Database; the schema has been done by Reuters' staff, so I have just created the stored procedures, views and functions that the application uses to manipulate the data.

One important detail: this application has been completely written "by hand", from scratch; I have not used Visual Interdev to create it (I just don't want to use it) and it exploits VBScript and JavaScript object-oriented capabilities at its maximum… which seems like saying hard work for little benefit right? Well actually the idea behind this design principle is to give Reuters an application that would be easily portable to .NET in the future. "Classic" ASP is a dead technology that will no longer be supported by Microsoft in the future…

The different modules have been separated in layers following a classic 3-tier architecture, and I used the following open source utilities to help me build it:

- Classic ASP Framework[1] by Christian Calderon;
- JavaScript Vector Graphics Library[2] and
- JavaScript Drag & Drop Library[3] both by Walter Zorn
- ASPUnit Unit Test Framework[4]

Here's some statistics showing different checkpoints during the month of September; these have been done with Campwood Software SourceMonitor Version 2.0[5] (nice utility, quite awkward, but hey, it's free :)

Date	Lines (SLOC)	# of files	Cumulated working days	Average lines per day
5 Sep	1075	20	2	537.5
12 Sep	1437	25	5	287.4
15 Sep	2745	31	8	343.1
19 Sep	4379	38	10	437.9
22 Sep	5245	45	13	403.5
25 Sep	6070	56	15	404.7
26 Sep	6235	58	16	389.7

Some observations:

- I have counted only the lines of the files that I have written (for those that might say that I've counted the lines of the open source frameworks ;)
- Yes, those figures include the source code comments. I have yet to find a tool that only counts "logical" SLOC; in other words, these numbers reflect "physical" SLOC
- Very little code has been generated by third-party tools in this application (something that modern IDEs do all the time); the good news is that the Classic ASP Framework[6] allows **incredible productivity** because of the high quality of the components and the ease of use… not to mention the fact that the API looks really similar to .NET… and that changes everything!
- The productivity jump from September 12th has to do with a particularly difficult point of the application that had to do with a

[1] http://www.claspdev.com
[2] http://www.walterzorn.com/jsgraphics/jsgraphics_e.htm
[3] http://www.walterzorn.com/dragdrop/dragdrop_e.htm
[4] http://aspunit.sourceforge.net/
[5] http://www.campwoodsw.com/sm20.html
[6] http://www.claspdev.com

graphical representation on the screen... once that was solved, the rest of the application “built itself” around that solution.

OK, hope that someone finds this information interesting... I do! It was real fun to do this application. I would love to show you more about it but, you know, this is propietary stuff and NDAs apply here :)

Thats What Namespaces Are For

Adrian Kosmaczewski

2005-09-29

Today I was looking at the list of speakers[1] in the Enterprise Architect Summit 2005[2] to be held in November in Barcelona... and found out that David Chappell[3] was one of them. Interesting thing, since I regularly read his blog articles about architecture and best practices. I even have one of his books[4]!

But... I looked at the photograph and said... "that's not David Chappell's photograph! They must have made a mistake when they did the page".

Well it turns out that the David Chappell whose blog I check frequently, is another one[5].

Actually, there are **two different people** called David Chappell down there, and to make things worse (or better!) both are speakers in worldwide conferences, both are american, both have written many books, and both are worldwide authorities on Service-Oriented Architectures.

How to distinguish them?

- **David A. Chappell**[6], also known as "Dave Chappell", is vice-president and CTO of Sonic Software[7]. He specializes in Java, and lives in the East Coast of the United States.
- **David Chappell**[8] (don't call him "Dave"), is Principal of Chappell & Associates[9]. He specializes in .NET, and lives in the West Coast of the United States.

As you can see, they are quite different in some regards. By the way, both have written articles about this coincidence! Here's one article[10],

[1] http://www.ftponline.com/conferences/eas/barcelona/speakers.aspx
[2] http://www.ftponline.com/conferences/eas/barcelona/
[3] http://www.ftponline.com/conferences/eas/barcelona/speakers.aspx#chappell
[4] http://www.amazon.com/exec/obidos/tg/detail/-/0596006756
[5] http://www.davidchappell.com/blog/
[6] http://www.oreillynet.com/pub/au/207
[7] http://www.sonicsoftware.com/
[8] http://www.davidchappell.com/blog/
[9] http://www.davidchappell.com/
[10] https://davidchappellopinari.blogspot.com/2004/01/on-two-david-chappells-problem.html

here's the other[11].

The funny thing is that Amazon just mixes the books from both authors[12] as if they were the same :)

PD: actually, here I found a third David Chappell[13] that, as it turns out, also works in the Computer field… and Dr. David Chappell[14] who's a history professor in the University of Arkansas. Well, there a surely lots of them, then?

Update, 2022-09-09: And now there's a comedian called Dave Chappelle[15] to make things even more confusing.

[11] https://web.archive.org/web/20030816174740/http://www.oreillynet.com/pub/wlg/2874

[12] http://www.amazon.com/exec/obidos/search-handle-url/index=books&field-author-exact=David%20Chappell&rank=-relevance%2C%2Bavailability%2C-daterank/103-5778019-6095842

[13] http://shakti.trincoll.edu/~chappell/

[14] http://www.uark.edu/depts/histinfo/history/chappell/Chappellindex.htm

[15] https://en.wikipedia.org/wiki/Dave_Chappelle

Propagandas

Adrian Kosmaczewski

2005-10-02

La publicidad suiza es de lejos la peor del mundo. Tiene cierta logica que sea asi, si uno piensa que es un pais con tal nivel de consumo que ni siquiera es necesario promover productos; se venden solos. La gente compra. Es una buena manera de definir el suizo como un consumidor; el suizo consume constantemente, freneticamente, es avaro, vive en abundancia de cosas que generalmente tienen reputacion de buena calidad, son caras pero en realidad no valen tanto. Pero el suizo esta dispuesto a pagar el precio.

Entonces, como el suizo esta dispuesto y preparado para onerosas expensas, la publicidad se vuelve ridicula, banal y hasta pone incomodos a los propios suizos, para decirles si es pesima. Es muy comun que en los cines, justo antes de la pelicula, pasen un par de anuncios publicitarios. En ese preciso instante uno se da cuenta de la reaccion de la masa a la publicidad: toses, risitas, manos en la frente, una sensacion de incomodo generalizada.

Una publicidad suiza se puede resumir a lo siguiente: una imagen alegorica. Tal imagen puede ser de dos tipos: inspirada de una obra de Kafka, con lo cual se vuelve incomprensible y ridiculamente complicada, o por el contrario, inspirada de una cancion de Pipo Pescador.

No hay termino medio; lo cual indica claramente que el suizo no ve su propio pais con ironia, sino con cierta imagen interna de “normalidad”. Uno no puede ni debe reirse de las cosas que son, ya que si “son”, es porque asi “debe ser”. Luego, al faltar ironia, la publicidad no puede generar sonrisas.

El efecto contrario se da en Argentina - casi diria yo, totalmente opuesto. El argentino es un critico profundo de su mundo (justamente ahi, muchas veces se va a la mierda). A partir de esa critica, surgen las ideas de las cuales se pueden establecer ridiculos o inconsistencias, que los productos promocionados pueden explotar para generar una sonrisa y asi convencer a un publico (poco propenso al gasto por varias razones) de comprar el producto.

Una vez mas: a partir del conflicto, surge el cambio. Este ultimo concepto es tal vez demasiado profundo para aplicarlo a la publicidad, pero la verdad es que es uno de esos patrones de funcionamiento generales que aparecen en diversos ambitos de la vida humana. En

Argentina el conflicto es constante; el cambio es indomito, y la publicidad se aprovecha de ello para obtener nuevas ideas. Puesto que la risa es un medio de canalizar la angustia generada por el cambio (entre otras cosas), la publicidad argentina usa la risa constantemente para demostrar lo ridiculo de una situacion y lo simple que seria si se usase tal producto. Una situacion similar se da en Francia (la television francesa tiene gran aceptacion en la parte francofona de Suiza), donde la idiosincrasia no es tan lejana de la argentina.

En Suiza, la falta de conflicto impide el cambio, luego la risa no existe, ya que no hay angustia (al menos no hay angustia con respecto al cambio intempestivo, prefiero aclarar). Al faltar la risa, la publicidad pierde su canal principal; solo le quedan las alegorias, que como dije se pueden agrupar en dos grandes sectores.

Sin embargo, esta falta de humor no la hace totalmente aburrida; el enfoque necesario es distinto. La publicidad suiza se concentra basicamente en los siguientes tipos de productos:

1. Productos farmaceuticos (de lejos en cabeza del peloton)
2. Seguros (vida, salud, etc)
3. Productos de lujo (relojes, viajes, autos de mas 40 mil dolares)
4. Chocolates, golosinas

Obviamente que durante ciertas epocas especiales (Navidad, dia de los enamorados, dia de la madre o del padre, etc) se hacen evidentes otras categorias tipicas como

5. Cosmeticos
6. Ropa

Asi, es raro ver una publicidades de productos alimenticios “normales”, tipo Aceite Marolio, mmm que olio, pilas Duracell con el conejito que siempre llega antes o mas rapido, etc. Las necesidades basicas estan cubiertas. Es “obvio” que se come, luego, para que promocionar la comida? Eso si: como es “obvio” que uno se va a morir, mas vale alargar el momento (medicamentos), tener algun seguro (como si la Zurich o la Rentenanstalt pudiesen transar con el diablo para que te venga a buscar recien a los 150 pirulos) o al menos vivir “bien” (tener un buen reloj, un buen auto, o al menos comerte un Snickers, etc).

Es un buen resumen de la vida por estos pagos.

My Bookshelf Part I

Adrian Kosmaczewski

2005-11-05

Working in the IT industry means learning continuously, and keeping lots of information in your brain at once; this, in turn, means subscribing to key newsletters, reading online forums and blogs, and last but not least, reading a bunch of books and PDF papers every year.

I try to read at least 6 books per year, covering some of the following subjects:

- Programming Languages, Platforms & Frameworks
- Software Architecture and Design
- Security
- Project Management and Methodologies
- Computing History and Industry Trends

Since I assume that if you are reading this, you work in IT, more particularly in software development, in this article I will give you a list of some books that I consider fundamental in my own career. That is, years worth of reading :) This is the first part of the article, giving the list of Programming Languages books; stay tuned for the next chapters covering the other topics.

The list is not exhaustive nor mandatory; I just consider it to give me a fine background for being productive and proficient in today's software development tasks. And while some of this books are old classics (one of them already more than 30 years old) the ever-changing IT environment will probably make this list become obsolete in less than 5 years. I think, however, that many of the titles listed below will remain as classics...

About the List

First I will explain what I mean when I say "6 books per year": believe me, I don't necessarily mean those 700 to 1000 pages long books with the name "Bible" on it; some very good IT-related books are less than 150 pages long, and still ground breaking.

And no, the "XXX for Dummies" nor the "YYY in 21 days" series books are to be taken into account. Period. They just do not compare to the O'Reilly or Addison-Wesley books.

And, please, do not try to convince me saying that you do not have time: as Joel says that writing is a muscle[1], I say that reading is another; the more you read, the easier it is to read faster and more accurately, and to retain what you just read; you surely know the frustrating feeling of not remembering the contents of the page before, because you just cannot focus your attention on the book.

I usually take 10 days to read a 200 pages book, but this average can change, mostly depending in whether I enjoy the book or not, and whether I have enough quiet moments during the week. For example, weekends are great moments to read, mostly under the sun, in my balcony or in the park near my home. You must find the right environment to read, and take the chance to relax and enjoy your book. Another great place for reading is the train; commuting from Lausanne to Geneva, every day back and forth, gives me lots of quiet moments to read (Swiss trains are something really out of this planet). The important thing is that you should enjoy reading what you read; if you don't stand a particular book, drop it, do something else, plug into your iPod, read Dr. Dobb's Journal, anything else. Don't force yourself to finish a book (unless you must because of some valid reason).

Another important thing about this list is that the order that I suggest does matter; even if you know most of the concepts in these books, some of them act as "frameworks" and are useful to get a stronger background on the subject. Lots of other books use these "framework books" as the base reference to expand the subject and present new viewpoints.

This is, actually, how I built my bookshelf: in a backwards manner; I jumped from one to the other, reading the references at the end of the books. **The rule of thumb for creating your own list is: if you see the same title being referenced in a lot of books, then buy it. You will not regret it.** Another useful way to find interesting books is to read the user comments in Amazon[2]; they are often a good reference for me.

For each referenced book, I will give the corresponding Amazon link, which contains the ISBN number, and if it has a related website, I will add the link as well (back in the 90's they used to include a CD with source code or software, and even before they used to carry a floppy disk...).

Index

- Books about Programming Languages, Platforms & Frameworks
- Books about Software Architecture and Design, and Security[3]

[1] http://www.joelonsoftware.com/articles/fog0000000036.html

[2] http://www.amazon.com

[3] /blog/my-bookshelf-part-ii/

- Books about Project Management and Methodologies, and Computing History and Industry Trends[4]

1. Programming Languages, Platforms & Frameworks

JavaScript Definitive Guide

by David Flanagan (ISBN 0596000480) http://www.oreilly.com/catalog/jscript4/ http://www.amazon.com/exec/obidos/tg/detail/-/0596000480

O'Reilly books are, in my opinion, among the best on the market; this one is no exception. If you do serious web development, don't think twice about it; buy this book right now. It is the absolute reference of the JavaScript language, with an excellent theoretical introduction to the language, and a deep reference of the different versions of the DOM as well as of the "core" JavaScript language. A definitive guide, but definitely not for beginners; a solid understanding of OO concepts is needed.

As a side note, I used it as the basis of a programming course that I taught last year, and the results were excellent. The problem with JavaScript is that it is the "World's Most Misunderstood Programming Language" as Douglas Crockford says[5]. This book shows clearly that JavaScript is an extremely powerful object-oriented language.

Applied .NET Framework Programming

by Jeffrey Richter (ISBN 0735614229) http://www.amazon.com/exec/obidos/tg/detail/-/0735614229/

If you really want to know how does the .NET Framework works, then this is the book you are looking for. The title is misleading; actually the book delves into the inner workings of the Framework and its most particular features (CLR, threads, delegates, attributes), helping developers understand better how to make better, faster, more secure applications. It is an absolutely required reading for any .NET developer.

Inside C# Second Edition

by Archer Whitechapel (ISBN 0735616485) http://www.amazon.com/exec/obidos/tg/detail/-/0735616485/

The perfect complement to Richter's book; this is not only a book about C#, but also a complete description of why C# is the best choice among .NET programming languages for nearly every project. Every

[4] /blog/my-bookshelf-part-iii/

[5] http://www.crockford.com/javascript/javascript.html

feature of the language is described in great detail, including the IL code generated by the compiler, explaining drawbacks and advantages. It must be said that this book is about C# 1.0, and as such it does not cover recent developments such as Generics. Other than that, the book is utterly excellent.

Learning Cocoa with Objective-C

by James Duncan Davidson (ISBN 0596003013) http://www.oreilly.com/catalog/learncocoa2/ http://www.amazon.com/exec/obidos/tg/detail/-/0596003013/

Another O'Reilly gem. I bought this book shortly after buying my first iBook in 2002, and I found it extremely easy to read, filled with short and clear examples. Cocoa is to Mac OS X what .NET is to Windows: a complete OO framework that allows to develop applications extremely quickly; but the comparison ends there: Cocoa is based in a dynamic typed language (Objective-C), similar to Smalltalk and Ruby, which makes everything soooo much easy to do.

James Duncan Davidson is a skilled Cocoa developer and consultant, and his blog is worth a read: http://x180.net/

Programming in Objective-C

by Stephen Kochan (ISBN 0672325861) http://www.amazon.com/exec/obidos/tg/detail/-/0672325861/

After reading Davidson's book, I wanted to know more about Cocoa; that's why I read Kochan's book to get a complete overview of Objective-C, which is with Ruby one of my favourite languages. This book does not show Objective-C in the context of Cocoa, but also in other implementations as well; actually, it takes and inverted approach to other Objective-C books. The author does not begin teaching C and then jumping into Objective-C, but rather teaches first Objective-C first, and deal with C afterwards. The journey is rewarding; you get to learn two languages for the price of one! And both with simple, concise examples.

Cocoa Programming

by Scott Anguish, Erik M. Buck, Donald A. Yacktman (ISBN 0672322307) http://www.cocoaprogramming.net/ http://www.amazon.com/exec/obidos/tg/detail/-/0672322307/

Well, I admit, I liked Cocoa so much that I got this book… and I do not regret it. Even given the size (nearly 1300 pages) it is an incredible reading, full of examples, details, insider tips, you name it. Each and every feature of Cocoa (as of Mac OS X 10.2 Jaguar, it must be said) is described here. Unfortunately there has not been an update to this book for Panther nor Tiger, so that new features (Core Data, Cocoa

Bindings, etc) are left to discovery in the Apple Developer Connection website.

Professional SQL Server 2000 Programming

by Rob Vieira (ISBN 1861004486) http://www.amazon.com/exec/obidos/tg/detail/-/1861004486/

Do you use Microsoft SQL Server 2000? You need this book. You must have it. 1400 pages worth of incredible details about the database engine, queries optimization, language features, you name it. It is a fundamental reference, and actually, the only Wrox book that I found worth buying (I tend not to like their books... well, actually they went bankrupt, so I think I was not the only one with such opinion...). Maybe they should change the picture in the front cover, the book may have better sales ;)

Coming soon

I will soon post more about my bookshelf...! Please do not hesitate to leave any comments... I would love to hear about you.

Puntualidad

Adrian Kosmaczewski

2005-11-13

En todos las estaciones de tren de Suiza, todas y cada una, sin excepcion, desde Zurich hasta Ginebra pasando por Lausana, Berna o Maienfelden (el pueblito de Heidi) existen unos relojes con cuadrante blanco, sin cifras ni numeros romanos, que indican todos exactamente la misma hora.

Tal vez me diran que dicho asi, esto parece una boludez, despues de todo este es el pais de los relojes; pero yo les digo, que es exactamente la misma hora, en todo el pais, con una precision de un segundo. Es asi; cuando el segundero de un reloj en Zurich indica que el minuto se termino y que son las dos de la tarde con cero minutos en punto, lo mismo sucede en cualquiera de las miles de estaciones de tren de los Ferrocarriles Federales Suizos, al unisono.

Es bastante simple como funciona esto. Los relojes estan coordinados (todos ellos) por una señal proveniente de dios sabe donde. Los segunderos dan la vuelta completa del cuadrante en solamente 57 segundos (es raro verlo en funcionamiento, puesto que van mas rapido de lo que deben, en realidad); los tres segundos restantes se quedan esperando la señal; cuando la señal llega, el minutero avanza un minuto y el segundero vuelve a empezar su carrera. Las agujas de los minutos avanzan de a saltos, no de manera continua como sucede habitualmente, mientras que la de los segundos avanzan de manera continua, y no de a saltos, como sucede habitualmente.

Pero igual, los trenes suelen tener retraso de todas maneras. Y la gente se enoja.

My Bookshelf Part II

Adrian Kosmaczewski

2005-11-13

This is the second part of the article "My Bookshelf"[1], with the list of the books that I recommend anyone in the software engineering field to read.

This second part has my preferred books about:

- Software Architecture and Design
- Security

As usual, all book references include links to their official websites, to Amazon.com as well as their ISBN, cover and list of authors.

Index

1. Books about Programming Languages, Platforms & Frameworks[2]
2. Books about Software Architecture and Design, and Security
3. Books about Project Management and Methodologies, and Computing History and Industry Trends[3]

2. Software Architecture and Design

In this specific area, the books from Addison Wesley[4] are by far the reference. But O'Reilly is catching up...

Object Oriented Analysis and Design with Applications, 2nd Edition

by Grady Booch (ISBN 0805353402) http://www.awprofessional.com/title/0805353402 http://www.amazon.com/exec/obidos/tg/detail/-/0805353402

This book is simply a mind-opening one. It is precisely this book who paved my way to my object-oriented skills. Whatever the programming language you work with (at the time it was VBScript, Transact-SQL and JavaScript) you will never develop software the same way

[1] /blog/my-bookshelf-part-i/
[2] /blog/my-bookshelf-part-i/
[3] /blog/my-bookshelf-part-iii/
[4] http://www.awprofessional.com/

after you've been through this book. It features a complete introduction to systems theory, giving full background on the object-oriented paradigm, its outcome and possibilities. The samples are in C++ (which I had to learn to fully get the ideas), but other than that, this book is a must have. It seems that there will be soon a third updated version, with samples in Java... so watch out for it.

By the way, the author is one of the creators of UML.

Design Patterns ("The GoF Book")

by Erich Gamma, Richard Helm, Ralph Johnson and John Vlissides (ISBN 0201633612) http://www.awprofessional.com/title/0201633612 http://www.amazon.com/exec/obidos/tg/detail/-/0201633612

Well, if there is just one book that you must read, is this one. You just cannot avoid it. Commonly known as the "GoF" ("Gang of Four") book, it is widely referenced by nearly all the other books, and it paved the basis for the software engineering way of this decade. The book features nearly 20 different common software design patterns, from the most obvious (Singleton, Adapter) to the most complex ones (Flyweight, Strategy, Observer), categorized in three categories: Creational, Structural and Behavioral patterns. If Object-Oriented Programming changed the way you develop software, believe me: Design Patterns will revolutionize the way you do it.

Head First Design Patterns

by Elisabeth Freeman, Eric Freeman, Bert Bates and Kathy Sierra (ISBN 0596007124) http://www.oreilly.com/catalog/hfdesignpat/ http://www.amazon.com/exec/obidos/tg/detail/-/0596007124

I found this book to be not only a fun and easy read, but also a welcome complement to the GoF book. The most common patterns (not all of them though) are described in great detail, with a complete explanation of the situations in which they make sense, highlighting the possible performance implications and common misconceptions. The samples are in Java and UML, and some paragraphs are frankly hilarious. Don't miss this one.

The Unified Modeling Language User Guide

by Grady Booch, James Rumbaugh and Ivar Jacobson (ISBN 0201571684) http://www.awprofessional.com/title/0321267974 http://www.amazon.com/exec/obidos/tg/detail/-/0201571684

UML is here to stay; you find UML diagrams in every book, press article or software specification document. It has become the "de facto" lingua franca of the software engineering industry (well, all but Microsoft, even if they implicitly use it without aknowledging it, as usual). This book (also known as the "Three Amigos" book) is the complete

reference to UML, with descriptions and examples of all the different diagram types, and their elements. A must have for anyone working in the industry.

MDA Explained - The Model Driven Architecture: Practice and Promise

by Anneke Kleppe, Jos Warmer, and Wim Bast (ISBN 032119442X) http://www.awprofessional.com/title/032119442X http://www.amazon.com/exec/obidos/tg/detail/-/032119442X

You cannot learn UML without seeing an immediate correlation between UML and "pure" source code. While many different attempts have been made to automate software development using visual tools, none has had the level of standardization and industry support of MDA. The OMG has established MDA as a standard specification, which vendors as well as the open-source community use now to create new tools, that are slowly getting more and more used in the industry. This book is a small but good introduction to the subject, highly recommended to software developers and architects.

Aspect-Oriented Software Development with Use Cases

by Ivar Jacobson and Pan-Wei Ng (ISBN 0321268881) http://www.awprofessional.com/title/0321268881 http://www.amazon.com/exec/obidos/tg/detail/-/0321268881

Aspect-Oriented Sofware Development (AOSD), also known as Aspect Oriented Programming (AOP) seems to be a paradigm that is really gaining momentum, and that could ultimately redefine software engineering in the next 10 years. This book, co-written by another of the creators of UML, gives not only a complete introduction to the subject of AOSD, but provides also a complete methodology for finding, designing and documenting aspects in UML. It is a complex book, I had to read it twice to really get it, it's extremely comprehensive from all points of view, and can be a tough reading. But not to be missed in any case.

3. Security

Hacking Exposed, Second Edition

by Joel Scambray, Stuart McClure and George Kurtz (ISBN 0072127481) http://www.hackingexposed.com/ http://www.amazon.com/exec/obidos/tg/detail/-/0072127481

Now this book is at its fifth edition! I got this book following the advice of a computer security teacher while in university, and it was an extremely useful read. I could not stop myself and started to download utilities referenced in this book... using them I found lots of security holes in the servers of the company I was working on... Also in the

book you get the explanation of firewall or DNS problems and tweaks, encryption, cryptography, common intrusion techniques, threat levels, believe me, this book has it all. For Unix or Windows, you are sure to find a solution for every single problem you might have. I must say that it made me become aware of how insecure Windows is… actually I became somewhat paranoid :)

Writing Secure Code, Second Edition

by Michael Howard and David LeBlanc (ISBN 0735617228) http://www.microsoft.com/mspress/books/5957.asp http://www.amazon.com/exec/obidos/tg/detail/-/0735617228

Oddly enough, Microsoft issued this book. And it is quite a good one if you develop applications for the Windows platform. However, I wonder whether all of the Microsoft staff read it (or understood it). Microsoft products continue to suffer from important security problems (Windows the first) but all in all this book gives, after a good introduction on the buffer overrun problem, lots of insight on code security, whether in .NET, COM or MFC programming. Definitive read.

My Bookshelf Part III

Adrian Kosmaczewski

2005-11-20

This is the third and last part of the article “My Bookshelf”, with the list of the books that I recommend anyone in the software engineering field to read.

This last part has my preferred books about:

- Project Management and Methodologies
- Computing History and Industry Trends

As usual, all book references include links to their official websites, to Amazon.com as well as their ISBN, cover and list of authors.

Index

1. Books about Programming Languages, Platforms & Frameworks[1]
2. Books about Software Architecture and Design, and Security[2]
3. Books about Project Management and Methodologies, and Computing History and Industry Trends

4. Project Management and Methodologies

The previous categories had more to do with technology; this section has to do with people, and the way that they interact with technology. I think that these books (at least the first two) should be mandatory readings in any computer science course.

The Mythical Man-Month: Essays on Software Engineering

by Frederick P. Brooks, Jr. (ISBN 0201835959) http://www.amazon.com/gp/product/0201835959/

This book is considered as one of the most fundamental, and indeed it is one of the finest books ever written on the subject of software project management; as Brooks once joked, “They call this book the Bible of Software Engineering... and that’s because everybody reads it but nobody does anything about it!”. Brooks was project manager of the ill-fated OS/360 project in the sixties, after having worked in IBM

[1] /blog/my-bookshelf-part-i/
[2] /blog/my-bookshelf-part-ii/

since the mid fifties, and wrote this book in 1974. It makes a comprehensive review of all the problems that plagued his project, describing them in great detail and providing solutions and best practices. It is an enlightening, short, easily readable and enjoyable book that is referenced almost everywhere. Heck, it even has its own Wikipedia entry[3]...

In 1995 this book was re-edited as a Special 20th Anniversary edition, augmented with two more chapters written by Brooks; one of them, No Silver Bullet is a controversial paper that has caused turmoil in the industry, and countless responses to its main statement: "there will be no more silver bullets, i.e., there will be no more technologies or practices that will create a 10-fold improvement in software engineering productivity over 10 years".

I cannot stress this too much; you are not a real project manager if you have not read this book (and applied its principles in your everyday work). Period. The sad thing of this is that none of my past project managers had even heard about this book, and even worse, most of them repeated the same mistakes that Brooks described. Why is not this book part of the official curriculum in universities?

Peopleware: Productive Projects and Teams

by Tom Demarco & Timothy Lister (ISBN 0932633439) http://www.amazon.com/gp/product/0932633439/

This one is another little gem; referenced almost everywhere, Peopleware is the result of almost ten years of research of two human resources consultants in New York. They have studied, from 1976 to 1986, hundreds of different software development companies, their human resources policies, the way they treat employees, the way they distribute office space, and condensed all of this information into a small but extremely valuable book, that provides a lot of pragmatic insight on how to build a successful software development team.

And most important, they make an underrated statement against open spaces. They say (and I fully agree with them) that open spaces are the #1 cause of failure of software projects. The lack of dedicated work spaces for programmers result in late deliveries, bad quality, high turnover, schedule slipages, and countless other problems.

Sadly (again) none of the HR guys Ive worked for has ever read this book (no exceptions), as is the case of all the project managers Ive had so far. I do believe that both The Mythical Man-Month and Peopleware should be the basis of any software management university degree.

[3] http://en.wikipedia.org/wiki/The_Mythical_Man-Month

Joel on Software: And on Diverse and Occasionally Related Matters That Will Prove of Interest to Software Developers, Designers, and Managers, and to Those Who, Whether by Good Fortune or Ill Luck, Work with Them in Some Capacity

by Joel Spolsky (ISBN 1590593898) http://www.amazon.com/gp/product/1590593898/ http://www.joelonsoftware.com/BuytheBooks.html

It all started as a blog on software engineering. Joel worked as a project manager of the Excel team back at the beginning of the nineties, then worked for an internet startup called Juno and after that, he had enough of managers who had never read neither The Mythical Man Month nor Peopleware (and for more reasons than that, actually), and decided to star his own business in New York: Fog Creek Software[4].

During that time, he wrote in his blog[5] his insight on building a software company, on how to manage software teams, how to hire and deal with people, all in all a very deep and practical knowledge; in 2004, he merged his best articles into this excellent book that I highly recommend. Joels point of view is that of a doer; no big management theories, no deep knowledge needed beforehand; practical tips, spanning from project schedule to office space set-up. Things that you will find useful whether you work for small company or a multinational.

Facts and Fallacies of Software Engineering

by Robert L. Glass (ISBN 0321117425) http://www.amazon.com/gp/product/0321117425/

This is another treasure; Glass makes a summary in this book of 65 principles, 55 facts (such as “Software estimates are rarely corrected as the project proceeds”, “Modification of reused code is particularly error-prone” or “Better methods lead to more maintenance, not less”) and 10 fallacies (“You can’t manage what you can’t measure”, “Software needs more methodologies”, “Given enough eyeballs, all bugs are shallow”), that surround the task of software engineering; he destroys common urban myths, show misconceptions and rectifies common notions that many managers take for granted. Short book that I just recommend to any project manager.

Software Project Survival Guide

by Steve C. McConnell (ISBN 1572316217) http://www.construx.com/survivalguide/ http://www.amazon.com/gp/product/1572316217/

This book and the next are both targeted to developers that have been promoted to the rank of project managers; this one, written by the author of Code Complete (a classic in my to-read list) gives useful

[4] http://www.fogcreek.com/

[5] http://www.joelonsoftware.com

insight in whats needed to achieve small projects; it gives a complete methodology (without any fancy name) targeted directly to not-so-large software projects. The descriptions are clear and concise, the book is well structured and it gives a good overview, and a good introduction, to the problems of project management and to how to solve them.

Leading a Software Development Team

by Richard Whitehead (ISBN 0201675269) http://www.amazon.com/gp/product/0201675269/

This one is the perfect complement to Software Project Survival Guide; also targeted to developers that suddenly face management tasks, Whitehead structured this book around 40 particular questions, one per chapter, so that the reader can randomly pick the answer for a particular question, without having to read the whole book. “I’ve just been made team leader of a new project, where do I start?”, “How can I stop my project from coming in late?” and “I’ve got someone on my team who’s a real problem, what should I do?” are some of the problems that the author tries to solve, and I have found so far that the answers provided are really pertinent.

5. Computing History and Industry Trends

Knowing the past helps build the future; personally I love to read the stories of how some ideas that we now consider for granted were created. It makes me wonder about the future developments, and helps me find trends in my job that would not be easy to spot otherwise. And, last but not least, it makes one understand that technology evolves quicker than human relationships do.

The Success of Open Source

by Steven Weber (ISBN 0674012925) http://www.hup.harvard.edu/catalog/WEBSUC.html http://www.amazon.com/gp/product/0674012925

This book is my 2005 preferred one. I must admit not having yet directly collaborated in an open source project (even if I have myself added features to some open source code for some project of mine), so this book gave me a good introduction (both historical, philosophical and economic) to the open source movement.

The author starts the book with the history of UNIX and its derivatives, stating clearly the political struggles among the different people involved in it, the economic outcomes, the licensing and copyright problems, and even the vocabulary needed to understand the different concepts. This book does not directly target technologists; actually the author himself states very clearly at the beginning that he is a professor of economic politics, not a technologist.

I think that all CIOs that continue to spend millions in propietary software, and avoid the open source solutions mainly because they do not have support should read this book and have a brighter idea about it. There is a whole world of opportunities behind the open source hype, and many projects out there (Linux, Apache, OpenOffice, GIMP, to name a few) are providing real solutions and working software for common problems. For free, and with the support of thousands of knowledgeable developers that have the same problems as you (instead of a phone support number that is always busy).

Dealers of Lightning : Xerox PARC and the Dawn of the Computer Age

by Michael A. Hiltzik (ISBN 0887309895) http://www.harpercollins.com/global_scripts/product_catalog/book_xml.asp?isbn=0887309895 http://www.amazon.com/gp/product/0887309895

At this precise moment, you are reading this website on a screen plenty of windows, menus and icons that you drive with a mouse attached to your computer. You most probably access this website with an Ethernet adapter, and maybe you even use a laser printer to get some hardcopies of all of this babble. Did you know that the mouse, the icons, the Ethernet and the laser printer (as well as Smalltalk and Object-Oriented Programming, Postscript, much later Aspect-Oriented Programming, and many other inventions) all come from the very same place, the Xerox PARC (Palo Alto Research Center) in Silicon Valley?

This book is a must, one of the best ever Ive ever read, all categories considered. If you want to know how all of these things were created, who are the incredible people behind these inventions, just buy this book. It is written as a novel, describing each character and every story in detail, and I really enjoyed it a lot.

Revolution in The Valley: The Insanely Great Story of How the Mac Was Made

by Andy Hertzfeld (ISBN 0596007191) http://www.amazon.com/gp/product/0596007191/ http://www.oreilly.com/catalog/revolution/

I am a Mac user, and this book is a great one for anyone interested in knowing how the Mac was created, between 1978 and 1984. Andy Hertzfeld joined Apple as an Apple][engineer, and he managed to get into the Macintosh team early on. His book is plenty of insight, details, and more than anything, an incredible amount of stories about the first Macintosh team. After reading this, you will wish you were born twenty years before in southern California...

CODE: The Hidden Language of Computer Hardware and Software

by Charles Petzold (ISBN 073560505X) http://www.amazon.com/gp/product/073560505X[6] http://www.charlespetzold.com/code/

This books gives a deep overview to many different ways used in the last two centuries to encode and transmit information, from the Morse code to programming languages used in current computers. It is an easily readable, enjoyable book, with plenty of information about the history of computers.

6. Coming Soon

These are the IT books that I am currently reading, or that I am planning to read soon:

From The Pragmatic Programmer[7]:

- Programming Ruby[8]
- Agile Web Development with Rails[9]
- My Job Went To India (And All I Got Was This Lousy Book)[10]
- The Pragmatic Programmer: From Journeyman to Master[11]

(actually I'm currently reading the first three of the above list)

From O'Reilly[12]:

- The Cathedral and the Bazaar[13]
- Better, Faster, Lighter Java[14]
- Beyond Java[15]

I will post details about these books as soon as I read them :)

Conclusion

I hope that this list will be a good starting point for those interested in software development; it is of course incomplete: there must be lots of books that you could consider fundamental and that do not appear here. That is why I would love to have your feedback about my list, and recommend me other titles as well! I would love to have new reading ideas from you.

[6] http://www.amazon.com/gp/product/073560505X/
[7] http://www.pragmaticprogrammer.com/
[8] http://www.pragmaticprogrammer.com/ruby/downloads/ruby-install.html
[9] http://www.pragmaticprogrammer.com/titles/rails/index.html
[10] http://www.pragmaticprogrammer.com/titles/mjwti/index.html
[11] http://www.pragmaticprogrammer.com//ppbook/index.shtml
[12] http://www.oreilly.com/
[13] http://www.oreilly.com/catalog/cathbazpaper/
[14] http://www.oreilly.com/catalog/bfljava/
[15] http://www.oreilly.com/catalog/beyondjava/

By the way, I've just found Joel's book reviews[16]...

Update, 2005-11-23: Joel has published another book list[17], this time targeted to the management training program of his company...

[16] http://www.joelonsoftware.com/navLinks/fog0000000262.html
[17] http://www.joelonsoftware.com/articles/FogCreekMBACurriculum.html

How to Install Ruby on Rails in Windows 2003

Adrian Kosmaczewski

2005-11-25

Great day; my first full Ruby on Rails[1] application is up and running in production environment. It is installed in an intranet server at my employer's premises, so I will not be able to show it :(

But anyway, here goes some tips and tricks for installing such an application in a Windows 2003 server, using a MySQL 5.0[2] database, and running it with Apache 2.0[3] with FastCGI[4] (pretty latest stuff altogether, isn't it??? :)

First of all, check this awesome article[5] and follow the instructions (here goes a local copy[6], if the URL does not work). That article gives almost everything you need to put the whole thing to run; this one just gives some details that I found useful this afternoon.

The most important, the packages to install (inspired from the above blog entry[7]):

- Install Apache2[8], latest stable version
- Install MySQL 5.0 Windows Essentials[9], latest stable version (download the client utilities as well, they are really useful)
- Install the latest Ruby Installer for Windows[10]
- Install Rails using RubyGems (`gem install rails --include-dependencies`)
- Install the latest Ruby For Apache[11]

The last one includes FastCGI and the Ruby-MySQL libraries. Beware though, that in my case I had an odd dialog box saying "xxxxx.DLL does not exist in C:\WINDOWS - Abort - Retry - Skip"; I just clicked

[1] http://www.rubyonrails.com
[2] http://www.mysql.com/
[3] http://www.apache.org/
[4] http://www.fastcgi.com/
[5] http://dema.ruby.com.br/articles/2005/08/23/taming-fastcgi-apache2-on-windows
[6] Taming_FastCGI_Apache2_on_Windows.pdf
[7] http://dema.ruby.com.br/articles/2005/08/23/taming-fastcgi-apache2-on-windows
[8] http://httpd.apache.org/download.cgi
[9] http://dev.mysql.com/downloads/mysql/5.0.html
[10] http://rubyinstaller.rubyforge.org/wiki/wiki.pl
[11] http://rubyforge.org/projects/rubyforapache/

on “skip” and everything went fine… maybe it has to do with the fact that this was Windows 2003… who knows!

Once you have installed these packages, the fun begins ;) Configuration time!

Apache’s httpd.conf file (in the conf subdirectory of the local Apache installation)

I assume here that your application is in “C:/path_to_your_rails_app”:

Remember **NOT to use backslashes** but common *nix slashes… Also, do not use quotes in your PATH variables; also, **use the 8.3 filenames for the PATHs** as well (you can retrieve them using “dir /X” in the command line).

Another important thing: if your application uses HTTP Authentication, add the ” -pass-header Authorization” parameter to the FastCgiServer directive, otherwise, you will not be able to retrieve the credentials of the logged in user in your application code…

Remember to restart Apache after every modification to httpd.conf!

In your /rails_app/public/.htaccess file

Change this line from

```
RewriteRule ^(.*)$ dispatch.cgi [QSA,L]
```

to this:

```
RewriteRule ^(.*)$ dispatch.fcgi [QSA,L]
```

so that you use FastCGI instead of “classic” (and slow) CGI.

The odd, undocumented, hidden trick

Once you’ve done all of this, guess what; **your application will NOT work**.

To fix it, you must do the following: go to C:\mysql\bin (or the bin file of your MySQL installation) and you will see a single DLL file (the name is something like libmysql.dll, I do not remember right now; but there is only one DLL file). Well, for the application to run, copy this file and paste it into C:\WINDOWS. Exactly, just do this, otherwise you will have a awful “Uninitialized constant mysql” error…

This last trick, It appears in comment 13 in the blog entry previously referenced[12], but in my case it worked if I copied it into C:\WINDOWS instead of C:\WINDOWS\SYSTEM32…

[12] http://dema.ruby.com.br/articles/2005/08/23/taming-fastcgi-apache2-on-windows

Restart Apache, browse to your web server, and voilà! Hope this helps!

Microsoft Support en Español

Adrian Kosmaczewski

2005-12-01

Sin comentarios.

http://support.microsoft.com/default.aspx?scid=kb;es;247970

Cómo para habilitar través, autenticación FTP UNC directorio virtual

> AVISO: Gracias por utilizar el servicio de Traducción Automática. Este artículo ha sido traducido por un sistema informático sin ayuda humana (Machine Translation). Microsoft ofrece estos artículos a los usuarios que no comprendan el inglés, exclusivamente, con el fin de que puedan entenderlos más fácilmente. **Microsoft no se hace responsable de la calidad lingüística de las traducciones ni de la calidad técnica de los contenidos de los artículos así como tampoco de cualesquiera problemas, directos o indirectos, que pudieran surgir como consecuencia de su utilización por los lectores.**

No, si ya nos dimos cuenta.

Ubuntu

Adrian Kosmaczewski

2005-12-04

This is my first post from a brand new world.

I have just installed Ubuntu 5.10 (“Breezy”)[1] in my G3 iBook, and so far I just love it. Fast, stable, incredibly easy to set up. The whole install took 45 minutes (10 minutes copying stuff from the CD to the hard disk, and then 35 minutes configuring itself). Ubuntu recognized every bit of hardware of this machine, from the Airport card to the trackpad and the Ethernet adapter. It comes with OpenOffice, GIMP and much more pre-installed, and I really find it easy to use. Very intuitive user interface, really great operating system. And free and open source.

Now I’m following the instructions in http://www.fo64.com/articles/2005/10/20/rails-on-breezy for installing Ruby on Rails on this brave new system.

Well the only problem found so far with Ubuntu is that the computer hang when I sent it to sleep (closing the lid of the iBook). When re-opening the lid, the computer would become unresponsive.

This post gives the exact procedure to do to solve it, until a fix comes in the next Ubuntu kernel...

http://ubuntuforums.org/archive/index.php/t-2368.html[2]

Only thing: in the scripts, replace “dbus-1” by “dbus” both in the name and the contents. This way it worked without problems on my G3 800 Mhz iBook...

I really look forward to use Ubuntu in my everyday work. I plan to use it as the default system for this laptop for a while. Stay tuned..

[1] http://www.ubuntulinux.org/

[2] https://web.archive.org/web/20051025192024/https://ubuntuforums.org/archive/index.php/t-2368.html

Radrails

Adrian Kosmaczewski

2005-12-10

After having received a comment[1] from Steven Ross[2] I saw in his website a reference to RadRails[3].

I must say that I have never used Eclipse before, but this application literally blew my mind. I had known it before doing the application I've blogged about before![4] I strongly recommend it: ruby code highlighting, integration with Subversion, available for Mac, Linux and Windows, data manipulation screens, and under strong development. The first version is dated October 2005 and apparently is getting lots of contributions; the current version is 0.5.1 and it is stable enough for everyday use.

Thanks Steven for the reference!

[1] /blog/how-to-install-ruby-on-rails-in-windows-2003/
[2] https://web.archive.org/web/20051211043147/http://www.zerium.com/zerium/
[3] https://web.archive.org/web/20051210025337/http://www.radrails.org/
[4] /blog/how-to-install-ruby-on-rails-in-windows-2003/

The Technical News of the Day

Adrian Kosmaczewski

2005-12-13

Ruby on Rails goes 1.0; read the announcement![1]

[1] https://web.archive.org/web/20051216094610/https://article.gmane.org/gmane.comp.lang.ruby.rails/34705

Gracias, y feliz vida, no solo para el 2006

Adrian Kosmaczewski

2006-01-30

2005 habra sido el año que espere durante mucho tiempo. Nada quedo en el tintero; nada quedo sin desvelar, ninguna bronca quedo sin el abrazo o el mate y ninguna caricia paso desapercibida. Nada, nada, nada. Y no solamente en mi vida, creo yo. El 2005 habra sido una bisagra fundamental y extraña donde se dieron vuelta muchas cosas. El que quiera ver, que vea.

Me siento raro, pleno y contento. Clau, el piano, la quena, España, el blog, el Master, Ruby on Rails, Bruselas, el laburo, la arquitectura, hasta el tobillo de la vieja son elementos de un año que paso literalmente volando como el huracan Katrina, llevandose todo por delante y escurriendose entre mis manos, dejandolas perfumadas con rocio y transpiracion, con perfumes lejanos y cercanos, agrios y dulces.

La vida, si cabe personificarla, es un gordito sonriente, al que poco le importan las formalidades. No es hipocrita, es fiestero, sabe lo que hace, y juega con los tiempos y los espacios que se nos alquilan durante el tiempo de nuestro paso en esta tierra. Juega con nosotros, y nos da la eleccion, como un grandulon que molesta a un petiso; o bien lo aceptas y te reis conmigo, porque no voy a parar, o pasate la vida (justamente...) llorando.

Y el 2006 se presenta aun mejor: Claudia, el Master, Libertango y Getronics aparecen en el horizonte como promesas de crecimiento. Promesas que me hago a mi mismo, despues de todo, para mostrarme que soy capaz de seguir creciendo un poco cada dia.

Lo mas hermoso de todo esto es que mi crecimiento ayude a otros. Poder ser catalizador de cambio. Ir por mas y sentirme acompañado en el camino.

No estar mas solo. El otro dia toque al piano “Libertango” con Pepe, acompañando su bandoneon; no fue un duo muy feliz, puesto que mis capacidades en el piano no estan aun muy pulidas, pero la sensacion de plenitud no me la saca nadie. Empiezo a entender lo que dicen los musicos sobre esa conjuncion que significa la musica de a muchos. Es algo enorme, que no tiene par.

Crecer y aprender mas. Este enero empece un Master online en cien-

cias de la computacion, algo que hace rato que quiero hacer. Va a ser una cantidad de trabajo considerable, pero la sensacion de poder dar tal paso es simplemente maravillosa.

Cambiar. Getronics, como lo habran deducido, es mi nuevo empleador. Thales no es mas. Aunque mas que Thales, casi podria decir que mi empleador era Nestle, ya que en 2 años trabaje en 4 proyectos .NET distintos dentro de la multinacional alimentaria. Dos como desarrollador, dos como arquitecto.

El espiritu de cambio ya estaba claramente instalado en mi, y ahora se autocumple la profecia (parafraseando mi amigo Fede).

Getronics merece un parrafo aparte. Es una empresa mas pequeña que Thales, si bien tambien es un grupo internacional de cierta relevancia, hasta ahora especializados en telefonica IP mediante routers Cisco, y que estan abriendose al campo del desarrollo de software .NET. Pero lo que me intereso en Getronics fue el espiritu que vi en la gente que encontre hasta ahora. Thales esta ahogandose en un management deficiente, siguiendo direcciones erroneas y sin prestar atencion a sus “colaboradores”. Hipocresia y mediocridad estan a la orden del dia, constantemente, en cada decision. Ya habia escrito sobre Thales[1]. No cabe decir nada mas.

Recuerdo una entrevista de trabajo en el Groupe SQLi, hace unos meses, con el gerente de la sucursal de Lausana. El tipo me pregunto “para usted, que es lo mas importante en un proyecto de software?”; y mi respuesta fue simple: “la gente”. El tipo no estuvo para nada de acuerdo, y con un aire sobrador y dictatorial me interrumpio y me dijo “No. Lo importante es la metodologia. Sin metodologia no hay software. La gente puede ser reemplazada”. Asi nomas, sin verguenza y sin mayor reflexion. Como correspondia a semejante cortesia, me levante, le di la mano y me fui.

Espero que Getronics sepa interpretar que lo importante es la gente. Antes de Getronics pase por 5 empresas, 2 en Argentina y 3 en Suiza. Y confimo lo que digo: lo importante es la gente.

Lo humano. Eso es lo que debe definir no solamente el software, sino cada elemento de esta vida.

Ahora bien, un consejo nomas; para el 2006, te recomiendo pensar bien fuerte, preguntandole a tu corazon lo que queres para el año nuevo. No se lo cuentes a nadie; pero desealo tan fuerte que sientas que tu pecho explota. Los deseos se cumplen, pero no esos, sino los otros. Los de verdad. Hay que leer la letra chica de los cuentos de hadas; ahi esta la explicacion. Lo demas es para la gilada. y creeme, funciona.

Que el 2006 nos sirva para definir lo humano, comenzando por nosotros mismos. Mil gracias por estar ahi, leyendo estas lineas.

[1] /blog/que-queres-que-te-diga/

Adios Nonino, Tal Vez

Adrian Kosmaczewski

2006-02-03

Raro destino el del abuelo de Piazzolla; tal vez la mayor razon que tuvo su ser, mas alla de sus logros personales, haya sido el de trascender para siempre en forma de cancion. Adios Nonino.

Tal vez nuestra unica razon de existencia sea el mero ser. Tal vez esa sea la clave de muchos de nosotros, el trascender de maneras no previstas, no sabidas, no queridas incluso. Tal vez esa sea la clave del por que la busqueda de gloria sea algo tan vano, tan ridiculo, que no tenga mayor razon de ser.

Tal vez el trascender sea funcion del azar. Tal vez la unica existencia sea la vincular, la del vinculo atorrante.

Tal vez no seamos si no hay otro para amarnos.

Starting a New Adventure

Adrian Kosmaczewski

2006-02-21

So here we go again.

Last week I started my new job, as Software Solutions Consultant for Getronics[1], in Lonay[2], near Lausanne[3] (where I live), Switzerland. Yeah, the commune of Lonay has an awful website. Anyway. The good news is Getronics, actually.

My job, as usual, will be one with multiple faces. I will be in charge of technical consulting for the pre-sales and development teams, and also architecture and training. As you can see, nothing to make you feel bored. But in any case, a big improvement over my previous employer[4], mostly in terms of growing possibilities, infrastructure, diversity of clients, and learning opportunities.

So my first two assignments go well in this scope: to begin with, a Visual Studio 2005 and .NET 2.0 customized development training for a big industry group in Switzerland, and also my first project as a project manager! A small development project for the Swiss public sector, that needed some redirection and some advice, and my boss has chosen me to manage it.

And more is coming in the next weeks… stay tuned! I cannot confirm yet but I have great news to announce soon.

So as you can see, many things going on simultaneously; at the same time I'm well into my Master of Science in Information Technology (MSc in IT, Software Engineering)[5], and managing again the website of the Martin Ennals Award[6].

The year 2006 has started at full throttle! :)

[1] http://www.getronics.com/
[2] http://www.lonay.ch/
[3] http://www.lausanne.ch/
[4] http://www.thales-is.ch/
[5] http://www.liv.ac.uk/study/lifelong_learning/distance_learning.htm
[6] http://www.martinennalsaward.org/

Get the Facts - I Mean, Get Them

Adrian Kosmaczewski

2006-02-26

If you enjoy Microsoft PR material, you may find this “Get the facts” page[1] somewhat interesting. For those of us who really deal with MS **and** Open Source stuff day by day, please just don’t laugh too loud.

My CV says I’m a “.NET blah blah blah”, and I’ve spent most of my professional life using and deploying MS technology. But you know what? I’ve had too many headaches with it. There’s always a gotcha; you cannot rely in their technology to do things more complicated that what the “getting started” demos say at first glance. Their APIs are often not completely implemented, and you cannot modify them if needed. You have to find workarounds to do things that you need, because their support website says that they won’t fix that problem until the next service pack, or worse, until the next version. And so on.

For example, in .NET 2.0 you can serialize DataTable instances into XML natively; nice. You can then use them (even if it’s not the best practice) as the result of a web service call. Cool. But did you know that the WSDL.EXE utility included in .NET 2.0 does not handle DataTable as return type for a web service? This is not documented either (at least I haven’t found it), so that the proxy generated by WSDL.EXE is completely flawed and you don’t know why your application does not work… until you Google on it. And since you cannot change it nor fix it, you are stuck to find another alternative.

They have a great contradiction between their approach to first-time users and hardcore development. There’s a mismatch; it just does not work the way they say it should.

So at that point, their comparison of Linux and Windows Server 2003 just does not make it any more. I do not buy it. I do not doubt that Windows Server 2003 might be a great product, as well as .NET, BizTalk Server or even SQL Server 2005, but the thing is, that none of the companies I’ve worked so far fully understood the licencing terms, and often just used the software without licence at all. And this makes everyone nervous.

[1] https://web.archive.org/web/20051026140504/http://www.microsoft.com/windowsserversystem/facts/default.mspx

It's just a pain; remember MSDE (Microsoft Data Engine), the free version of SQL Server 2000? Well for example you just could not deploy it for your own applications BUT following a specific set of licence rules, one of which was the fact of "not using it in replacement of Microsoft Access"; that is, if Access could be used in the same context (how come?) then MSDE could not be used. The result was a good deal of applications and websites using Access for data storage instead of MSDE (!). I know a few.

Result: lock-down, imposibility to migrate data to other systems (well, yes, there's a couple of SourceForge projects allowing you to migrate but... why aren't these native features?) and overall unhappyness of developers and maintainers.

I know that there's not only the problem of ease of use or licencing; what about performance? What about ease of setup? MS was the first, it's true, to offer real drag-n-drop installation of compiled web applications; but now they are not the only ones, and their solution seems older and more complex compared to others[2]. The Open Source world is growing today at incredible speeds, and innovation and new developments are happening before in the Open Source space rather than in the proprietary one.

So what? Are we gonna stick to ASP.NET just because there's a whole bunch of project managers out there that think that it's the only way to do things? Just because they do not know (and sometimes they do not want to know) that there's a life out there? It's like pretending that SAP might be the answer for everything.

There is no silver bullet[3].

Those who say, today, that using open-source technology is hard, is just because they haven't used any of it. At all. And poor them, they just do not know what they are missing.. For a complete roadmap of technologies, just check this article of mine[4]; it might help to find your way. And grab an old PC and download and install Ubuntu[5]. You might as well be surprised. And drop that copy of MS Office altogether; OpenOffice.org[6] is just plain great stuff.

There's a fight going on. If you don't believe me just read this article[7].

[2] http://www.rubyonrails.com

[3] https://web.archive.org/web/20040713074912/www.virtualschool.edu/mon/SoftwareEngineering/BrooksNoSilverBullet.html

[4] /blog/state-of-the-art/

[5] http://www.ubuntu.com

[6] http://www.openoffice.org

[7] https://web.archive.org/web/20060307200143/https://searchopensource.techtarget.com/columnItem/0,294698,sid39_gci1165420,00.html?track=NL-301&ad=541985

Wow!

Adrian Kosmaczewski

2006-02-28

Apple has published an article about Ruby on Rails[1] in the Developer Connection website!

[1] https://web.archive.org/web/20061017073930/https://developer.apple.com/tools/rubyonrails.html

What Will the Software Architecture Discipline Look Like in 10 Years' Time

Adrian Kosmaczewski

2006-03-16

This is a tricky question; after all, Bill Gates himself published a book in 1995, "The Road Ahead", where he only slightly talks about the World Wide Web:

> "The Road Ahead" appeared in December 1995, just as Gates was unveiling Microsoft's master plan to "embrace and extend" the Internet. Yet the book's first edition, with its clunky accompanying CD-ROM, mentioned the Web a mere seven times in nearly 300 pages. Though later editions tried to correct this gaffe, "The Road Ahead" remains a landmark of bad techno-punditry – and a time-capsule illustration of just how easily captains of industry can miss a tidal wave that's about to engulf them. (Salon.com, 2000)

Thus, trying to extrapolate our own craftsmanship up to 2016 is inherently tricky, but a nice thought experiment after all.

Software Architecture, today

First of all, I should say that I work primarily as Lead Software Developer and (lately) as Software Architect. My clients are businesses that need to automate some of their day-to-day tasks, to achieve better productivity and lower costs. This already narrows the type of applications I work in, to the good old three-tiered application, usually on top of a database. Lately this applications began to communicate a lot more between them, raising a (higher level) concept known as Service Oriented Architectures.

My objectives, as an architect, are the following:

- Creating strong, dynamic and knowledgeable teams
- Lowering development and maintenance costs
- Creating value by continuous innovation and research

I think that the role of the Software Architect will strengthen in the future; but it is my hope that we will become less astronauts (Joel

Spolsky, 2001), that we will mix up with developers (junior and senior) and that we will keep a close eye into the real, final product that software is made of: source code. It is very easy to fall into the trap of losing the context, and spend too much time talking about buzzwords and trends:

> Now it's tagging and folksonomies and syndication, and we're all supposed to fall in line with the theory that cool new stuff like Google Maps, Wikipedia, and Del.icio.us are somehow bigger than the sum of their parts. The Long Tail! Attention Economy! Creative Commons! Peer production! Web 2.0! (Joel Spolsky, 2005)

The core substance of software deserves more eyes and more minds, thinking ways to describe not only the big picture (something that you can do with fancy diagrams) but also to give solutions to the problems that developers find daily while building systems up. Software is a process, but not any kind of process: a human one, maybe the most intangible of all processes; and as such, it is filled with all human brightnesses and failures.

The Future

There are several aspects that, in my opinion, will be key to understand the future of my activity; I will go one by one, explaining them (or providing pointers for more information) and giving some ideas about the benefits they would bring.

1) Aspect-Oriented Programming (AOP)

Object-Oriented Programming OOP took 20 years to get mainstream (from Simula in the late 60s, to the creation of Smalltalk in the XEROX PARC in the mid 70s, to Java in 1995); I think that AOP (actually, AOP was also created by a XEROX PARC scientist) will have a similar, if no longer, incubation time; this is because of its relative complexity.

AOP aims to complement OOP orthogonally:

> Aspect orientation is a set of technologies aimed at providing better separation of crosscutting concerns.

(Ivar Jacobson, 2004)

An archetypical example of crosscutting concern is logging; usually logging code is scattered throughout business application code, which ultimately makes maintenance harder; what if a bug is discovered in a third-party logging component, and hundreds or thousands of method calls must be changed? Other examples of crosscutting concerns are configuration, security, resource management or error handling. Using AOP, the business application code can be completely separated from crosscutting concerns; these are "mixed" ("weaved", using AOP

terminology) during runtime following specific conditions, in specific execution points (“pointcuts”, as they are called in AOP).

I think that AOP will be a key tool in the future, helping architects model better systems, helping developers concentrate in key business rules, and lowering the costs of maintenance. One big drawback of AOP nowadays is execution speed, since runtime weaving usually is a costly operation:

> I tested three cases: the generation of the Trace.WriteLine call through AOP.NET, the Trace.WriteLine call manual coded before the call to the empty virtual function, and the Trace.WriteLine call without the virtual function call. The result was that using AOP.NET to cross-cut the logging was eight times slower than manually coding it. Again, these results are for cross-cutting a light-weight logging call onto an empty virtual method. For a fatter cross-cut onto a fatter method, the relative impact of the cross-cut will be much smaller. My test case is close to a worse-case scenario, so don’t ditch AOP.NET or AOP in general based on this result. On the same token, don’t be dismissive of the impact of doing compile-time-like things at runtime. The cost can be high. (Nick Wienholt, 2005)

2) Design Patterns

“Design Patterns” is a concept taken from the world of “real” architecture, used to describe certain conceptual elements that hold integrity and offer style to a building. In the case of software, the term Design Patterns is closely related to one of the most important software-related books written in the 90s: Design Patterns: Elements of Reusable Object-Oriented Software (Addison-Wesley Professional Computing Series, ISBN 0-201-63361-2) (http://www.amazon.com/gp/product/0201633612/103-2097252-2747802). This book identifies almost 20 common OOP designs, useful for solving common problems found in software development. This book laid the path for other similar books, describing common networking, business or enterprise patterns.

I think that in the future, design patterns will be incorporated in programming languages (Ruby already defines Singleton, Observer and Iterator directly in its syntax - see RubyGarden in References), in visual development tools and in the overall toolset of architects and developers. In other words, the overall level of abstraction will raise, and design patterns will be used to describe these new levels.

3) Inversion of Control Frameworks

One of the most interesting design patterns is what Martin Fowler names “Inversion of Control”:

> Inversion of Control is a common phenomenon that you come across when extending frameworks. Indeed it's often seen as a defining characteristic of a framework. (…) One important characteristic of a framework is that the methods defined by the user to tailor the framework will often be called from within the framework itself, rather than from the user's application code. The framework often plays the role of the main program in coordinating and sequencing application activity. This inversion of control gives frameworks the power to serve as extensible skeletons. The methods supplied by the user tailor the generic algorithms defined in the framework for a particular application. (Martin Fowler, 2005)

This pattern or characteristic of is being widely exploited by a growing number of frameworks or "containers" that accelerate considerably the development of applications:

- Spring (http://www.springframework.org/ and http://www.springframework.net/)
- Hibernate and other persistence frameworks (http://www.hibernate.org)
- ASP.NET (http://www.asp.net)
- Struts (http://struts.apache.org/)
- Cocoa (http://developer.apple.com/cocoa/)
- Ruby on Rails (http://www.rubyonrails.com/)

All the packages named above provide not only pre-built functionality in their libraries, but also (and most important) overall application scaffolding, sequencing and instrumentation; they tightly define how an application will behave, they provide configuration options (so that the behavior can be changed later without modifying the source code) and also they provide standardization and portability, in the sense that usually the code that sits on top of these frameworks can be ported to other platforms in the future.

While these frameworks arguably reduce the creativity of the developers (and usually this is a complaint) or could impact the overall performance of the final application (something that is less of a problem every new release), the benefits that these packages bring greatly outcome the drawbacks, mostly regarding the shorter delivery times and the lower maintenance costs.

4) Dynamic and strongly-typed programming languages

This topic could likely be the one to raise more religious wars in the next 10 years; all developers have their preferred programming languages, and usually they tend to defend them against all odds, whenever possible, touting their advantages.

While the benefits of strong-typing in enterprise applications are clear (compile-time bug detection, ease of maintenance, clear syntax), the

value dynamic languages can provide is yet to be seen, Java and C# (both statically typed) being the most widely used languages in these environments. Nevertheless, Ruby on Rails (http://www.rubyonrails.com) already provides an enterprise-class framework based in the Ruby language, and it is slowly gaining acceptance thanks to: speed of development, ease of maintenance, strong-typing and readability; the basic tradeoff between dynamic and static languages are the number of instructions needed to perform a task; the more dynamic the language, the less lines of code are needed:

> Vastly reduced code footprint : We have read our Fred Brooks and respect the fact that there has never been a "Silver Bullet", and there is unlikely to ever be such. Rails is not a Silver Bullet. However, widely reported results place productivity increases over modern Java methodologies (e.g., J2EE , Struts, etc.) in the 6-fold to 10-fold range (with many of these claims coming from long-time Java luminaries). (Rick Bradley, 2005)

I think that dynamic languages like Ruby will be more accepted in the future for enterprise development. A big drawback against wide acceptance today is the lack of proper editors (Statically typed languages benefit from IDEs like Eclipse and Visual Studio.NET, which provide syntax help during development time; this is harder, if not impossible, to do with dynamic languages such as Ruby).

5) Model-Driven Architectures (MDA) and code-generation techniques

Code-generation techniques are mainstream today. Several different commercial and open-source packages allow to generate source code, usually from a design built in some kind of modeling tool; lately one of the methods that has gained the biggest momentum is MDA (http://www.omg.org/mda/), which stands for Model Driven Architecture. MDA uses standard UML to define and design application from different perspectives, and defines a basic workflow that ultimately is translated into source code.

MDA enables, for example, to create the skeleton of the same application in several different platforms (hardware or software), for example, Java, .NET and Cocoa, or the Web and the Desktop, simultaneously. This way, one can achieve knowledge reuse at architectural level, bringing more productivity and adding more value.

I think that MDA will grow in acceptance and support (it already has a lot of support), and will ultimately become a useful standard in 10 years' time.

Conclusion

Another factor that I could have added in the list above is Open Source Software (OSS), but I think that this is a huge one that ultimately surrounds all the others without distinction, and gives them more strength and value (both social and economically speaking). OSS is a subject in itself, that would have made this article longer that it already is!

It will be interesting to re-read this article in 2016 and see how wrong I am now; I think that in Computer Science, 10 years are worth 10'000 in archaeological times: animal species appear and disappear, ice ages come and go, volcanoes explode and deserts replace forests. The whole landscape changes continuously, but I strongly think that the core of our activity, the code, will remain the center of our attention. Only time will tell if in 10 years' time we will still use Emacs to type our code, or not.

References

Bruce A. Tate, "Beyond Java", ISBN 0-596-10094-9, O'Reilly, 2005

Ivar Jacobson, Pan-Wei Ng, "Aspect-Oriented Software Development with Use Cases", ISBN 0-321-26888-1, Addison-Wesley, 2005.

Joel Spolsky, April 21st, 2001 [Internet], "Don't Let Architecture Astronauts Scare You", http://www.joelonsoftware.com/articles/fog0000000018.html (Accessed January 27th, 2006)

Joel Spolsky, October 21st, 2005 [Internet], "Architecture Astronauts are Back", http://www.joelonsoftware.com/items/2005/10/21.html (Accessed January 27th, 2006)

Martin Fowler, June 26th 2004, "Inversion of Control" [Internet] http://martinfowler.com/bliki/InversionOfControl.html (Accessed January 27th, 2006)

Nick Wienholt, "AOP Performance Numbers for .NET" [Internet], http://ablog.apress.com/?p=454 (Accessed January 27th, 2006)

Rick Bradley, "Evaluation: moving fromm Java to Ruby on Rails for the CenterNet rewrite" [Internet], http://rewrite.rickbradley.com/pages/moving_to_rails/ (Accessed January 27th, 2006)

RubyGarden, "Example Design Patterns In Ruby" [Internet], http://www.rubygarden.org/ruby?ExampleDesignPatternsInRuby (Accessed January 27th, 2006)

Salon.com, 2000 [Internet], "Why Bill Gates still doesn't get the Net", http://archive.salon.com/21st/books/ (Accessed January 27th, 2006)

A New Programming Language Every Year

Adrian Kosmaczewski

2006-03-29

Somewhere I read that it was a good thing to learn at least one new programming language every year; I think I have kept up that trend since 1992:

- 1992: QBasic[1]
- 1993: Turbo Pascal[2]
- 1994: C[3]
- 1995: Delphi[4]
- 1996: Java[5]
- 1997: JavaScript[6]
- 1998: VBScript[7]
- 1999: Transact-SQL[8]
- 2000: C#[9] and Prolog[10]
- 2001: C++[11]
- 2002: PHP[12]
- 2003: Objective-C[13]
- 2004: Visual Basic.NET[14]
- 2005: Ruby[15]

And this year's winner is: LINQ[16]. The main purpose of learning it is to prepare the LINQ conference in the TechDays next week... and this

[1] http://en.wikipedia.org/wiki/QBasic_programming_language
[2] http://bdn.borland.com/article/0,1410,20803,00.html
[3] http://en.wikipedia.org/wiki/C_programming_language
[4] http://en.wikipedia.org/wiki/Borland_Delphi
[5] http://java.sun.com/
[6] http://www.crockford.com/javascript/javascript.html
[7] http://msdn.microsoft.com/library/en-us/script56/html/0a8270d7-7d8f-4368-b2a7-065acb52fc54.asp
[8] http://msdn.microsoft.com/library/en-us/tsqlref/ts_tsqlcon_6lyk.asp?frame=true
[9] http://msdn.microsoft.com/vcsharp/
[10] http://en.wikipedia.org/wiki/Prolog
[11] http://www.cplusplus.com/
[12] http://en.wikipedia.org/wiki/PHP
[13] http://en.wikipedia.org/wiki/Objective-C
[14] http://msdn.microsoft.com/vbasic/
[15] http://www.ruby-lang.org
[16] http://msdn.microsoft.com/netframework/future/linq/

is huge indeed!

A More Boring World

Adrian Kosmaczewski

2006-04-02

While I was reading this blog post[1] about "Easter eggs" in Microsoft products, I came accross these two (utterly brilliant) comments in the same page:

The first from Kevin Daly[2]:

> Let's be honest, we all know perfectly well that the real reason corporate customers don't like Easter eggs is that in the core of their shrivelled little souls they believe that having fun while you work is the same as stealing from the company.
>
> This is the culture of the suit, the meaningless mission statement, and the ruthless elimination of all signs of joy and humanity.
>
> The Open Source movement's got it all wrong: it's not software that needs to be made free, but the people who use it. And the people who write it.

The second from mark[3]

> The word you're missing when discussing Easter Eggs is "PRIDE". Easter eggs wind up in software because someone is proud of their work. The original easter egg in a game was put in there because the programmer wasn't getting any credit anywhere in the code, on the box, or in the manual... and he was proud of what he'd done.
>
> You want software written by people who take pride in their work, the same way you want a car built by someone who isn't disgruntled at his employer or food served to you by someone who hates their customers.

Talking about pride...

Just for the sake of explanation, an "Easter egg" is a hidden piece of code, usually included in commercial shrinkwrap software, that shows some credit page, or some game, or some other funny stuff done by

[1] http://blogs.msdn.com/larryosterman/archive/2005/10/20/483110.aspx
[2] http://www.dotnetjunkies.com/weblog/kevdaly/
[3] http://www.hostile.org/blog/

the developers of the product during their spare time (more about that in Wikipedia[4])

It seems, though, that even if during the eighties and nineties, Microsoft products usually shipped with Easter eggs, "Nowadays, adding an easter egg to a Microsoft OS is immediate grounds for termination, so it's highly unlikely you'll ever see another." (source: Larry Osterman[5]).

That's why I found the two comments above to match my opinion on this subject. I do believe that Microsoft took this decision following (big) corporation comments, and that these corporate clients (I have seen them closely) do not have the slightest idea on how software is done, and what is the motivation of software developers. I talk here about the true developers, those that care about their products and try to make the best possible products; 9-to-5-lemmings do not count here (and there are a lot).

I think Easter Eggs are a fundamental part of the process, from a human point of view; don't take me wrong, I understand project managers and their fears of spending time in useless features; I understand testers and their concerns for Q&A, but I also understand the developers - I'm one of them, even if now I'm doing architecture and project management tasks.

A software development process that does not take social and human matters into account is useless.

Why can't developers have fun they way they intend to? **Why the only "teambuilding" solution found by software companies is boring meetings in some far away place where we are all "supposed" to have fun?** Why not having internal projects so that they can measure up their coding skills, and by the way, learn new things that could bring money to the company? Developers are not like anyone else. Specially the good ones, and I mean that word.

Software companies, at least those that I've seen in the Geneva Lake area, do not know what software means in the minds of those who do it. Because let's be clear, it's not the managers who type the code; it's the developers. If they're unhappy, your products will be crap and they'll leave the company sooner or later. Why don't management try to understand them, instead of blaming them for the bad quality?

The picture is simple: developers type code in unfriendly, noisy environments, usually in badly managed projects with little time or resources; while the CIO goes around the world in Business Class, higher project managers meet at the Hyatt for a business breakfast or lunch, and sales people go to bootcamps. Projects come out late, the company spends too much money, and the only solution is to yell at the developers. Why is that? I am beginning to think that managers are envious of the salary developers get. There must be something like

[4] http://en.wikipedia.org/wiki/Easter_egg_(virtual)

[5] http://blogs.msdn.com/larryosterman/archive/2005/10/20/483110.aspx

that. **You must never spit on the workers that do the job for you, for they are the most important part of your production chain, whatever the craft, whatever the profession.**

This cannot continue. I'm deeply unhappy of what's going on. And wherever I look, I find the same patterns and the same odd reactions.

Lugares Donde Todos Se Conocen

Adrian Kosmaczewski

2006-04-02

steven paul jobs nacio el 24 de febrero de 1955 en san francisco, california, de un doctor en ciencias politicas sirio (abdulfattah “john” jandali) y de una maestra de escuela (joanne carole schieble) que, no pudiendo o no queriendo tenerlo en su regazo, lo entregaron en adopcion a paul y clara jobs, abogados de la ciudad de mountain view, en el condado de santa clara, california.

joanne decidio hacerlo, segun steve declarara años mas tarde, porque queria asegurarse de que su hijo se recibiria en la universidad.

los padres de steve se casaron mas tarde y dieron nacimiento a la hermana menor de steve; los dos hermanos se conocerian de adultos.

steve, vegetariano y budista, nunca termino la universidad; es mas, fue expulsado a pesar de sus altas notas.

hace exactamente 30 años, el 1ro de abril de 1976, steve jobs fundaba apple computer junto a sus amigos steve wozniak y ronald wayne.

ronald wayne vendio su parte de apple computer a los dos steve dos semanas mas tarde, por 800 dolares.

la computadora “apple I”, hecha a mano por wozniak y jobs en el garage de la casa de paul y clara jobs, costaba 666.66 dolares; unos 200 ejemplares fueron construidos.

en 1977 steve jobs tuvo una hija con chris brennan, pero el no reconocio la niña sino años mas tarde. su nombre es lisa brennan-jobs.

en diciembre de 1980 apple entra en bolsa; su valuacion bursatil es la mas grande desde que ford entro en bolsa en 1956; apple vale 1’700 millones de dolares; de sus 1000 empleados, 40 son millonarios ipso facto.

la primera computadora de apple con mouse, menues y ventanas no fue el macintosh, sino otra, mucho mas cara y menos exitosa, llamada “lisa”.

la hermana de steve es hoy dia una escritora de cierto renombre, que escribio un par de bestsellers; su nombre es mona simpson, aunque no se sabe si es su nombre verdadero o su apodo artistico.

en 1985, steve es echado de apple; con parte de la plata que tenia, le compra a george lucas una parte de lucasfilm; la empresa se llama pixar, su costo es de 10 millones de dolares.

en la serie “los simpsons”, mona simpson es el nombre de la madre de homer simpson. la hija de homer simpson se llama lisa y es vegetariana y budista.

uno de los primeros dibujantes de “los simpsons”, brad bird, trabaja actualmente en pixar, y fue el director de “the incredibles”, exito de pixar en 2005.

mona simpson publico en 1995 una novela, “a regular guy” (“un tipo normal”) donde describe a un tipo que no termino la universidad, que se hace millonario en silicon valley, lanzando una empresa en el zotano de sus padres, mientras que tiene que manejar su relacion con una hija no reconocida.

en diciembre de 1996 steve jobs vuelve a apple; en 2006, steve vende pixar a disney por 7’400 millones dolares.

la verdadera mona simpson, hermana de steve jobs, vive en santa monica, california, con su marido, richard appel.

lisa brennan-jobs es tambien escritora.

richard appel, el marido de mona simpson, hermana de steve jobs, trabaja desde 1989 como productor ejecutivo y escritor de guiones de series de dibujos animados.

entre ellas, richard appel fue productor y guionista de “los simpsons”.

nota del 2006-04-29: steve jobs contrato a matt groening, creador de “los simpson”, para hacer un folleto o manual sobre la macintosh en 1989, justo antes de que salte a la fama: http://homepage.mac.com/mbishop/PhotoAlbum30.html

Hardware Polymorphism

Adrian Kosmaczewski

2006-04-08

Since data and instructions are stored in RAM in pretty much the same way, a priori the CPU cannot distinguish each other, but by the cycle in which the binary chunk is fetched from memory. In the case of instructions, it then needs to decode the operation codes into instructions, with the added problem that if the operation is performed on data that is not implied by the operation code, the results are wrong or even catastrophic.

The question is: would it be useful if in hardware, each cell of data would carry its own type designation? I will discuss here the pros and cons of this approach, in respect to hardware and software architectures.

Introduction

The example of the + sign is particularly interesting; here's an excerpt of a tutorial for the Ruby programming language, where the need for data types appears in a straightforward way:

> "Before we get any further, we should make sure we understand the difference between numbers and digits. 12 is a number, but '12' is a string of two digits.
>
> Let's play around with this for a while:
>
> ```
> puts  12  +  12
> puts '12' + '12'
> puts '12  +  12'
> ```
>
> (results)
>
> ```
> 24
> 1212
> 12  +  12
> ```
>
> How about this:
>
> ```
> puts  2  *  5
> puts '2' *  5
> puts '2  *  5'
> ```
>
> (results)

```
10
22222
2  *  5
```

(Chris Pine, 2006)

As we can see in the above example, higher-level programming languages allow us to distinguish (if not explicitly like Java, contextually like Ruby) among different types of information, whereas, at hardware level, this distinction does not exist; the processor executes or processes instructions or data depending on the processor cycle.

Metadata

In other words, if meaningful information (data) is stored in the memory of the computer and can be processed by a computer program, then we can say that every bit (no pun intended) of information has, at least at a certain abstraction level, and from a certain point of view, a particular type, or, more generally, some metadata attached to it:

> "Metadata (Greek: meta- + Latin: data"information"), literally "data about data", is information that describes another set of data. A common example is a library catalog card, which contains data about the contents and location of a book: It is data about the data in the book referred to by the card. Other common contents of metadata include the source or author of the described dataset, how it should be accessed, and its limitations."

(Wikipedia, 2006)

In our case, the type of a particular piece of data is part of its metadata:

> "Assigning datatypes ("typing") has the basic purpose of giving some semantic meaning to otherwise meaningless collections of bits."

(Wikipedia, 2006)

(Imaginary) Type-checking processor architecture

Now, let's suppose that a certain processor architecture allows us to distinguish in-memory pieces of data from in-memory program instructions. How could this be implemented?

First of all, let's see the different primitive types of information that a processor could distinguish:

- Integer numbers (of different but defined lengths such as 8, 16, 32 or 64 bits)
- Floating-point numbers (again, of different but defined lengths)

- Single characters (single- or multibyte-characters, such as Unicode ones)
- Strings (of variable lengths)
- Pure binary streams (images, audio, video)
- Uniform arrays or vectors (of variable length, where the items are all of the same type)
- Variable arrays or vectors (of variable length, where the individual items can be of any type, similar to C structures)

Let's imagine, to begin, that this is the definitive list of supported types at hardware level, by a certain microprocessor architecture. I have kept this list particularly close to that of any common high-level language, for reasons that will become obvious in a while.

How could the type metadata be stored at hardware level? The easiest way to imagine this is having a supplemental byte at the beginning of each in-memory variable or structure, indicating the type. This would give us the possibility of referencing 256 different types of data, which is more than enough in this particular example.

In the case of variable length data types as shown above (String, Arrays) another byte (or bytes) should indicate the length of the whole data structure. The need for this will be explained below.

Type-checking

Now, during the execution type, the processor would fetch data from memory but this time, it would have a first byte of information about the information (metadata) indicating the type of what follows, and eventually some length information as well. Instead of having to rely in context (which is the case by now), the processor could proactively check that the data will be processed by the appropriate instructions. This is a common technique used in programming languages called "Type Checking":

> "The process of verifying and enforcing the constraints of types - type checking - may occur either at compile-time (a static check) or run-time (a dynamic check). Static type-checking becomes a primary task of the semantic analysis carried out by a compiler. If a language enforces type rules strongly (that is, generally allowing only those automatic type conversions which do not lose information), one can refer to the process as strongly typed, if not, as weakly typed."

(Wikipedia, 2006)

In the case of a processor-based type check, it would be a pure strong, dynamic one.

Of course, this introduces the first drawback of this approach; while current processor architectures bypass this check and blindly trust

the “context coherence” between data and instruction, the processor would have to execute an internal check to verify them prior to executing the instruction. A smarter approach would be to have the processor to “trust” some executing code, if it comes from a statically-typed compiler, for example; this way, the processor would not execute the type check, and would have a similar behavior as that from current systems; however, it would execute a supplemental check for code coming from dynamically-typed languages (such as scripting languages).

Security

A direct benefit of type-checking processor architectures such as the one described above has to do with security. One of the most common security problems in software today is inherent to the Von Neumann architecture, in which data and instructions are both loaded in memory and share adjacent locations. This security problem is known as “Buffer Overrun” or “Stack Overrun”:

> “A stack-based buffer overrun occurs when a buffer declared on the stack is overwritten by copying data larger than the buffer. Variables declared on the stack are located next to the return address for the function’s caller. The usual culprit is unchecked user input passed to a function such as strcpy, and the result is that the return address for the function gets overwritten by an address chosen by the attacker. In a normal attack, the attacker can get a program with a buffer overrun to do something he considers useful, such as binding a command shell to the port of their choice”.

(Howard & LeBlanc, 2003, page 129)

(Howard & LeBlanc follow this statement with a C program that shows the security failure, and explain how it might be exploited to inject code in the computer program and change its behavior)

In the case of the stack buffer overrun, the problem is not only that current processors do not check the type of the data to process, but they do not even check the length of it. Length-checks should be then the first check that a processor should do before processing in-memory data of variable length (as stated above, Strings, Arrays and structures like C structs fall in this category).

This leads to infere that a type-checker processor would be particularly useful in security intensive environments (such as nuclear plants, life support systems, etc) where the tradeoff of performance for an additional security type check could be highly desirable.

Virtual Machines

Virtual machines such as Java or .NET’s Common Language Runtime (CLR) both perform length and type checks; in those cases, the virtual

machine specification ensure that the code being executed does not perform illegal memory accesses or reference objects of the wrong type at the wrong moment.

For example, .NET's CLR includes a runtime security engine that constantly checks code metadata, to know whether the method calls are trusted or not:

> "Permission demands propagate up the stack. When a method call demands a particular type of permission, the security engine must affirm that every component in the stack (prior to the point of the permission demand) has appropriate permissions. If any component does not, the permission demand fails and an exception is thrown to signify this failure. Each frame of the stack can modify the effective set of permissions by calling Assert, Deny or PermitOnly before making calls, and there are also calls to Revert changes made earlier. Taken together, this mechanism results in aggregate behavior that is constrained by the least privileged component that is participating in a given stack region"

(Stutz, Neward & Shilling, 2003, page 185)

This, among other reasons (such as automatic memory management) make virtual machines a "hot topic" in computing nowadays, since they allow to develop much more secure systems, with greater productivity, with fewer resources.

Of course, it must be said, not all security problems have disappeared with virtual machines, but that's another topic.

Other benefits

I think that another performance benefit could come from the fact of having native string manipulation at hardware level. String manipulation is by far the most common operation performed in high level programming languages (where Perl and Basic are the most common examples), but yet until now text strings as such exist only at that high level (and even until recently, when the Standard Type Library appeared, C++ did not even have a native string type - http://www.bgsu.edu/departments/compsci/docs/string.html).

Common string operations that could be implemented at hardware level include string copying, string concatenation and splitting; this way, getting the length or a defined substring of a given string would need a single processor instruction, instead of the current procedures, that imply memory allocation and copying, both extremely expensive in time and resources:

> "In all cases, these functions consist of copying all or a subset of a string to another string. The specific steps are: _

- Determine the number of characters to copy
- Allocate space for the characters
- Copy the characters to the new string Because of the memory allocation and copying operations involved, extracting sub-strings is also an expensive operation."

(VBIP.com, 2006)

Conclusion

The inclusion of high-level instructions in processors is not something new. It allows to boost the speed of hardware architectures, providing common operations to be performed at maximum speed at the lowest system level. One good example of existing implementations is the Velocity Engine existing in PowerPC G4 and G5 microprocessors:

> "The Velocity Engine, embodied in the G4 and G5 processors, expands the current PowerPC architecture through addition of a 128-bit vector execution unit that operates concurrently with existing integer and floating-point units. This provides for highly parallel operations, allowing for simultaneous execution of up to 16 operations in a single clock cycle. This new approach expands the processor's capabilities to concurrently address high-bandwidth data processing (such as streaming video) and the algorithmic intensive computations which today are handled off-chip by other devices, such as graphics, audio, and modem functions.The AltiVec instruction set allows operation on multiple bits within the 128-bit wide registers. This combination of new instructions, operation in parallel on multiple bits, and wider registers, provide speed enhancements of up to 30x on operations that are common in media processing"

(Apple Computer, 2006)

Some example code in C that uses the AltiVec instruction set is shown in http://developer.apple.com/hardware/ve/tutorial.html

Of course, these implementations greatly impact compiler and operating systems design, but they do not (I think) impact higher-level languages such as Java, C#, or scripting languages such as Perl or Ruby, who tend to be rather platform-independent (both software and hardware).

References

Apple Computer, "Velocity Engine" [Internet], http://developer.apple.com/hardware/ve/ (Accessed February 3rd, 2006)

Chris Pine, "Learn to Program" [Internet], http://pine.fm/LearnToProgram/?Chapter=02 (Accessed February 3rd, 2006)

David Stutz, Ted Neward & Geoff Shilling, “Shared Source CLI Essentials”, ISBN 0-596-00351-X, O’Reilly, 2003

Michael Howard & David LeBlanc, “Writing Secure Code, 2nd Edition”, ISBN 0-7356-1722-8, Microsoft Press, 2003

VBIP.com, “String Operations” [Internet], http://www.vbip.com/books/1861007302/chapter_7302_04.asp (Accessed February 3rd, 2006)

Wikipedia, “Datatype” [Internet], http://en.wikipedia.org/wiki/Datatype (Accessed February 3rd, 2006)

Wikipedia, “Metadata” [Internet], http://en.wikipedia.org/wiki/Metadata (Accessed February 3rd, 2006)

Bola De Nieve

Adrian Kosmaczewski

2006-04-17

Parece mentira, que en un pais como Suiza, la gente sea tan torpe sobre la nieve; bueno, si uno lo analiza detenidamente, no es tan raro. No nacimos para caminar sobre hielo, basicamente, y el hielo no es tampoco la mejor superficie para caminar o conducir un auto. Y la nieve, en ultima instancia, es hielo.

Es asi como de repente, un dia de invierno, sin avisar ni nada, el pais se cubre de blanco; te despertas a la mañana con 30 centimetros de nieve delante de la puerta de tu casa. Entonces, como Lausana esta construida sobre un monstruo desnivel sobre el lago, aprendes a esquiar sin quererlo; porque no caminas, patinas a lo mejor, asi que mas vale aprender a patinar. Y llegas a la parada del bondi, cubierto con tu gorro, tus guantes, tu bufanda y los tamangos con suela mas gorda pero igual te cagas de frio, porque a -10 grados es dificil no tener frio. Y el bondi no llega, obviamente, porque no puede circular a mas de 10 kilometros por hora…

Entonces uno estaria tentado y pensaria: bueno, pero no es la primera vez que sucede esto, la gente tendria que estar acostumbrada; y la verdad es que es imposible acostumbrarse a la nieve. Es asi como se oye luego en las noticias que hubo 50 accidentes en la autopista solamente durante la mañana.

About Operating Systems and Networks

Adrian Kosmaczewski

2006-04-17

> "The true operating system is the net itself"

This phrase, common marketing argument in the late nineties, made me remind that in the eighties, Sun Microsystems' founder, Scott McNealy, used the slogan "The Network is the Computer" to describe his vision.

But, can we safely mix both concepts?

Actually open your browser, type "The Network is the Computer" (with quotes) in Google, hit the "I'm Feeling Lucky" button and you will be redirected to Sun's website:

> "In 1982, we were just a group of young guys who believed The Network is the Computer. Today, I'm proud to say this principle still guides Sun's business and is driving many of today's prominent innovations, trends and business models. It's great to be back on stage with this group of big minds and talk about the history and evolution of Silicon Valley, the future of the network, and what this all means as we enter a new age of participation on the global network."

(Scott McNealy, 2006)

Now, somehow I cannot help myself of thinking that such a statement, actually is some kind of word game, where one mixes up different common but fairly unrelated concepts, and at the end you get a very good marketing phrase. And with all marketing phrases, they play mercilessly with ambiguous concepts:

> "Most current usage of the term"operating system" today, by both popular and professional sources, refers to all the software that is required in order for the user to manage the system and to run third-party application software for that system. That is, the common understanding includes not only the low-level "kernel" that interacts directly with the hardware, but also libraries required by applications as well as basic programs to manipulate files and configure the system.
>
> The exact delineation between the operating system and application software is not precise, however, and is occa-

sionally subject to controversy."

(Wikipedia, 2006)

From that point on, trouble (and marketing) begins; remember that Microsoft strategy of binding the internet browser to the operating system, making it impossible to be replaced, and taking it from application to system level. It not only changed the shape of the computer desktop, it also started lawsuits, concept breaks and countless reboots.

The key point for Sun's statement is that very limit between system and application software:

> "Let us begin by dividing a machine's software into two broad categories: application software and system software"

(Brookshear, 2004)

When this limit is fuzzy, one can take the whole concept of the network layer to the lower level of the system software, and thus Sun's catchphrase gets its meaning.

For me, one thing is a computer running an operating system (any brand or technology), and another very different is a set of computers connected through some kind of network. The first is an entity by itself, self-contained, that can run without a network connection; the latter exists from the moment that there are at least two nodes connected. There is an implicit dependency from the network to the computer. From that point of view, the "true" operating system was, is and will be the one running in the node.

There is, however, some trends and shared characteristics that show that a new "abstraction level" is appearing, as network-capable operating systems become common; computers connected in networks achieve much more than isolated ones. What are, then, the factors that contribute most to this development?

In my point of view, the most important of all these trends are:

- Open Source
- Standards and Interoperability

Each one of these trends make the line between operating system and networks each time fuzzier, thus creating the illusion that both can be merged in the same concept. Let's give a brief look at each one of them.

Open Source

I think that the Open Source movement shapes this thin line, providing strong competitors to commercial packages, stimulating creativity through software source code availability, and spanning new business models all over the landscape:

> "The key point about having source was that you could see how other people did things. This radically lowered the barriers to learning, and because learning by example means you don't have to spend your energy reinventing the wheel, imitation soon sparked innovation.
>
> We saw a similar explosion of creativity in the early days of the web. Tim Berners-Lee's original web implementation was not just open source, it was public domain. (...)
>
> But even more significantly, the "View Source" menu item migrated from Tim's original browser, to Mosaic, and then on to Netscape Navigator and MSIE. Though no one thinks of HTML as an open source technology (because of the fixation on licensing), it's been absolutely key to the explosive spread of the web. Barriers to entry for "amateurs" were low, because anyone could look "over the shoulder" of anyone else producing a web page. Dynamic content created with interpreted languages continued the trend towards transparency."

(Tim O'Reilly, 2000)

I think that what Tim says is brilliant, and I can only recommend reading the whole article. The capability of discovering how things work raises the quality bar of the systems, guarantees its availability in poor countries (since they just need a network connection to get them for free), thus "lowering the barriers of entry" and helping others contribute new ideas in the whole network, which then grows in value following Metcalfe's law:

> "Metcalfe's law states that the value of a network equals approximately the square of the number of users of the system (n ^ 2). Since a user cannot connect to itself, the actual calculation is the number of diagonals and sides in an n-gon"

(Wikipedia, 2006)

In terms of the thin line between network and operating systems, this means that newer ideas using network connections appear, later seamlessly incorporated in operating systems.

Standards and Interoperability

The trouble of connecting different computers, using different hardware and running different operating systems can be achieved by:

- Having a single company working to ensure that its products run in all target platforms using a particular propietary protocol (a rather expensive option nowadays, but the only option until the eighties)

- Relying in open standards, established by joint associations and used all over the industry.

The latter option, much more rational from an economic point of view, has led to the creation of a set of organizations that set standards used throughout the industry:

- IEEE Computer Society Portable Application Standards Committee (PASC) that establishes the POSIX standard for operating systems (http://www.pasc.org/plato/)
- Internet Engineering Task Force (IETF), http://www.ietf.org/
- World Wide Web Consortium (W3C), http://www.w3.org/

All of these organisms have members coming from industrial and academic backgrounds, and provide a common foundation, best practices and formal standards that allow the industry to go to new levels.

In the field of networking, this helps by providing standards such as SOAP (Simple Object Access Protocol, http://www.w3.org/TR/soap/) which allows computers in networks to securely share "information services", interchanging information in XML format (http://www.w3.org/XML/) using the HTTP protocol (http://www.w3.org/Protocols/).

This level of interoperability effectively turns networks to the level of operating systems, where the concept of low-level API gets a higher level meaning of "service".

Conclusion

I do not think that we can mix the concepts of operating system and network so easily; they should be always relied to their original meanings. There is, however, a synergy that comes from their combination and that makes the whole platform stronger than it was before.

References

J. Glenn Brookshear, "Computer Science, An Overview, Eighth Edition", ISBN 0-321-26971-3, Addison Wesley, 2005

Sun Microsystems, Scott McNealy, "Sun Microsystems' Founders Take Center Stage at the Computer History Museum" [Internet], January 11th, 2006, http://www.sun.com/smi/Press/sunflash/2006-01/sunflash.20060111.1.html (Accessed February 11th, 2006)

Teruo Koyanagi, "Photo Album Online" [Internet], http://www.csg.is.titech.ac.jp/~mich/photos/ (Accessed February 11th, 2006)

Tim O'Reilly, "The Network Really Is the Computer" [Internet], June 8th, 2000, http://www.oreillynet.com/pub/a/network/2000/06/09/java_keynote.html (Accessed February 11th, 2006)

Wikipedia, “Metcalfe’s law” [Internet], http://en.wikipedia.org/wiki/Metcalfe’s_law[1] (Accessed February 11th, 2006)

Wikipedia, “Operating system” [Internet], http://en.wikipedia.org/wiki/Operating_system (Accessed February 11th, 2006)

[1] http://en.wikipedia.org/wiki/Metcalfe\XeTeXglyph\numexpr\XeTeXcharglyph”0027\relax{}s_law

Primavera en Verde Y Azul

Adrian Kosmaczewski

2006-04-28

Tras seis meses de invierno, se destapa la naturaleza que ya no es tan virgen. Hablo de la zona del lago Leman, donde lindan los Alpes, el Rodano, Lausana, Ginebra, Evian, y una multitud de otras pequeñas ciudades repartidas entre Francia y Suiza.

Los colores son fuertes, encandilan, hacen mal al mirarlos fijamente: no queda otra, porque despues de tan feroz invierno, la naturaleza pega un grito de aburrimiento y agonia; el verde es rabioso, es violento; la profundidad del lago se vuelve azul, al fin, en vez de gris; las montañas dejan de ser blancas para volverse verdes y marrones, los vientos se aplacan, las temperaturas se vuelven mas dulces. Los arboles, finalmente, vuelven a serlo; en vez de tristes esqueletos de madera, se pueblan de retoños, pichones, ardillas y mariposas. El pasto se puebla de gotitas frias.

Es una epoca grandiosa. Justo ahora, delante de mis ojos.

La gente, en su voragine diaria, hasta llega a esbozar sonrisas, tanto el sol nos hace cosquillas. Las calles se pueblan de terrazas hasta altas horas de la noche (nueve, diez) donde se dejan volar hormonas, nunca se sabe.

La primavera es un grito, el de la reafirmacion de la vida; es un grito de victoria que se repetira hasta que algun meteorito o bomba atomica nos extirpe de este planeta verde y azul.

You’ve Got Mail!

Adrian Kosmaczewski

2006-04-28

Cuando AOL (America On Line) alla a fines de los 80, le puso a su sistema de correo electronico una voz de chabon que decia “You’ve got Mail!” cada vez que llegaba un nuevo mensaje, poco se imaginaba Ginebra de que diez años mas tarde hasta los gorriones tendrian direccion de e-mail.

Y lo mas comico es que ese desarrollo de la informatica de los 90 fue en gran parte el resultado indirecto de un invento de las afueras de Ginebra, en un laboratorio de fisica nuclear llamado CERN, alla por el 89; donde un cientifico britanico invento una manera de encadenar documentos en red, y asi nacio la Web.

El CERN esta en Meyrin, que es una comuna fronteriza con Francia, a unos 20 Km al oeste de la ciudad de Ginebra. Pero en plena ciudad, algo la predestinaba a tal rol.

Es asi como en Ginebra existe la “Avenue du Mail”. Mira que tener una avenida dedicada al e-mail parece pelotudo, pero en realidad no es asi; el asunto es que mas el tiempo pasa, menos se acuerda la gente del porque se llama asi esa avenida. O sea: que es un “Mail” en frances? Hoy dia, la respuesta es inmediata. Y hace 50 años?

Y como es una avenida importante, hay un edificio de la benemerita Universidad de Ginebra que se llama “Uni Mail”.

Y aun mejor: a unas pocas cuadras de ahi, hay una calle con un panel recordatorio indicando la extinta ocupacion de aquel cuyo nombre honra la calle en cuestion. El cartel, de cuya calle no recuerdo el nombre, reza: “Mengano (1500 y algo, 1600 y algo), peintre sur émail”.

El chabon era “pintor sobre esmalte”. Pero convengamos que es llamativo.

Kubuntu 5.10 and the Linksys WPC54GS Wireless-G Network Adapter

Adrian Kosmaczewski

2006-05-01

OK, so this time I've tried to make the same I've described before[1], but for Kubuntu 5.10[2]. I have changed to Kubuntu since I like KDE more than gnome, and also, Kubuntu seems to run faster than Ubuntu. And so I said to myself, OK, these are the same guys who make Ubuntu and Kubuntu, the wireless stuff should work fairly easily. After all it's the same kernel...

Wrong.

The steps to follow are these:

1. Follow these steps[3] from 1 to 7 (step 8 applies only to Ubuntu, not Kubuntu)
2. Edit the file `/etc/network/interfaces` (typically `sudo vim /etc/network/interfaces`) and add the following:

```
iface wlan0 inet static
pre-up ifconfig wlan0 up
pre-up ifconfig wlan0 down
pre-up ifconfig wlan0 up
pre-up ifconfig wlan0 down
pre-up iwconfig wlan0 essid YOUR_ESSID_HERE
pre-up iwconfig wlan0 mode Managed
pre-up iwconfig wlan0 key YOUR_NETWORK_KEY_HERE
pre-up ifconfig wlan0 up
wireless-essid YOUR_ESSID_HERE
address YOUR_CHOSEN_IP_HERE (192.168.1.50, for example)
netmask 255.255.255.255
gateway THE_IP_OF_YOUR_WIRELESS_ROUTER_HERE (192.168.1.2, for example)
```

(Thanks to this page[4] for the tip!)

[1] /blog/how-to-install-the-linksys-wpc54gs-wireless-g-network-adapter-in-ubuntu-5.10-breezy/

[2] http://www.kubuntu.org

[3] /blog/how-to-install-the-linksys-wpc54gs-wireless-g-network-adapter-in-ubuntu-5.10-breezy/

[4] http://rt2x00.serialmonkey.com/phpBB2/viewtopic.php?p=7335&sid=ee2e3c8aa62b8698d7531390c1d249d7

3. Type `sudo modprobe ndiswrapper` (if you haven't done it before)
4. Type `sudo /etc/init.d/networking restart`
5. Type `sudo ifconfig wlan0 up`
6. Type `sudo ifup wlan0`

Voila! You should be online by now. I have yet to find a way to execute the last 4 commands automatically when my computer boots up…

Geneva Techdays 2006 Powerpoint Slides

Adrian Kosmaczewski

2006-05-01

Well the PowerPoint slides that I used during the TechDays 2006 conference have been published in the TechDays page[1]. You can download them from this site as well:

- A204 - SharePoint Workflow[2] (zipped, 6.5 MB)
- D308 - LINQ[3] (zipped, 6.7 MB)

BTW, they are in French... and yes, the slide with Maradona on it is maybe the most important of them all :))

[1] http://www.microsoft.com/switzerland/techdays/fr/agenda.mspx
[2] A204.zip
[3] D308.zip

Quick Comparison of C Sharp and Ruby

Adrian Kosmaczewski

2006-05-05

I have been working as a software developer since 1996, and as such I've used a variety of different languages, both compiled and interpreted. But the who languages that I know and use most today, are two somewhat different ones, **C#** and **Ruby**. I will begin my presentation with a short explanation of both, providing their major similarities and differences, and then providing some code samples of both.

Both languages are ranked #7 and #21 respectively in the TIOBE Programming Community Index, as of February 2006 (http://www.tiobe.com/tpci.htm).

The C# Programming Language

C# was created by Microsoft as part of its .NET programming environment, and while not oficially acknowledged, it is deeply inspired in Java:

> "C# (pronounced"C sharp") is a simple, modern, object-oriented, and type-safe programming language. It will immediately be familiar to C and C++ programmers. C# combines the high productivity of Rapid Application Development (RAD) languages and the raw power of C++."

(Microsoft, 2000)

It was designed by Anders Hejlsberg, who also created the well-known programming environments of Turbo Pascal and Delphi, while he worked for Borland before joining Microsoft. Even if being a propietary language, its core elements have been standardized by the ECMA and the ISO standard bodies, respectively as the ECMA-334 and ISO/IEC 23270 standards.

Applications written in the C# language run inside the CLR, which is the name of the virtual machine environment; there are several implementations of the CLR, one being .NET itself, the other being the open-source Mono runtime (http://www.mono-project.com/)

C# is a compiled, strongly-typed, object-oriented language. In C#, everything is an object, unlike Java where, for example integer variables and other basic types are not objects per se. Memory management is done automatically by a "garbage collector" facility built into the CLR,

much like Java does. Actually, C# provides single inheritance with interface support, that is, exactly the same approach that Java provides. All these similarities, coupled with the syntax similarity, shows clearly the common design principles of both programming languages.

I have been using C# in my day-to-day job since 2002, and as such I've found the following features as the most useful:

- A particularly "picky" compiler: the C# compiler performs a high number of checkings which avoid common run-time problems related to type casting. In fact, C# does not allow all types of type casting to be done automatically, and the warnings provided by the compiler help the developer to create more stable and maintainable applications.
- Runtime capabilities: even being a language with a strong compiler and static typing, the .NET runtime is also capable of lots of operations by itself, mostly using reflection capabilities, and C# makes using these capabilities very easy. These are interesting but also dangerous, since they also can have a strong impact on performance.

The Ruby Programming Language

Ruby was created in 1993 by Yukihiro Matsumoto. It is an interpreted, dynamically typed, object-oriented language available in nearly all hardware and software platforms, for free; indeed, Ruby is an open-source programming language, for which its implementation can be freely used in any context, without restrictions of any kind.

Ruby has been inspired from Smalltalk, Perl and Python, and designed from the very beginning with the developer in mind:

Matz's primary design consideration is to make programmers happy by reducing the menial work they must do, following the principles of good user interface design. He stresses that systems design needs to emphasize human, rather than computer, needs: Often people, especially computer engineers, focus on the machines. They think, "By doing this, the machine will run faster. By doing this, the machine will run more effectively. By doing this, the machine will something something something." They are focusing on machines. But in fact we need to focus on humans, on how humans care about doing programming or operating the application of the machines. We are the masters. They are the slaves.

(Wikipedia, 2006)

Ruby also supports single inheritance, but instead of providing interfaces, it provides "mixins", which allow a class to use the functionality defined in modules without having to subclass them.

I started using Ruby last year (2005) because of the discovery of the overhyped application framework "Ruby on Rails" (http://www.rubyon

rails.com/) that allows the rapid development and deployment of web-based applications using the Ruby language. I had the opportunity to use Ruby on Rails in a couple of applications and had an unparalleled level of productivity with it.

Similarities and Differences

Like C#, Ruby is a pure object-oriented language; in both languages, everything is an object. Both languages use exception handling to notify errors, allow object introspection, both have automatic memory management through garbage collection and allow the creation of multithreaded applications. They are also very "hyped" languages, which have a huge level of support in the industry, by different vendors and communities.

The code below does exactly the same task in both languages: (Download files)[1]

In Ruby:

```ruby
# To execute, type "ruby employee.rb" at the command line
# (tested with Ruby 1.8.2)

class Employee
    def initialize(name, age)
        raise "Cannot be so young!" if age < 0
        raise "Cannot be so old!" unless age < 130
        @name, @age = name, age
    end

    def greet
        puts "Hi, my name is #{@name} and I am #{@age} years old"
    end
end

begin
    raise "Where are the parameters?" if ARGV.length < 2
    name, age = ARGV[0], ARGV[1].to_i
    employee = Employee.new(name, age)
    employee.greet
rescue
    puts "There was an error: " + $!
ensure
    puts "The program finished!"
end
```

In C#:

```
/*
To compile this program, just type
```

[1] employee.zip

```
"csc employee.cs" in your command prompt;
this will generate an "Employee.exe" in the same path
(tested with the .NET Framework 2.0)
*/

class Program
{
    static void Main(string[] args)
    {
        try
        {
            if (args.Length < 2)
            {
                throw new System.Exception("Where are the parameters?");
            }
            string name = args[0];
            int age = int.Parse(args[1]);
            Employee employee = new Employee(name, age);
            employee.Greet();
        }
        catch (System.Exception e)
        {
            System.Console.WriteLine("There was an error: {0}",
                                     e.Message);
        }
        finally
        {
            System.Console.WriteLine("The program finished!");
        }
    }
}

class Employee
{
    private string name = string.Empty;
    private int age = 0;

    public Employee(string name, int age)
    {
        if (age < 0)
        {
            throw new System.Exception("Cannot be so young!");
        }
        if (age &gt; 130)
        {
            throw new System.Exception("Cannot be so old!");
        }
        this.name = name;
        this.age = age;
    }
```

```
    public void Greet()
    {
        System.Console.WriteLine("Hi, my name is {0} and I am {1} years old",
            this.name, this.age);
    }
}
```

The most important difference among C# and Ruby is typing. Indeed, C# is a statically-typed language while Ruby is a dynamically-typed one. The dynamicity of Ruby has the advantage of making programs shorter and more readable, but has also the drawback of being a trap-door, causing hard-to-find run-time errors, specially when used by inexperienced developers. In the above example, the C# program will raise an exception if you specify a string instead of a number for the age... while the Ruby interpreter will not complain!

On the other side, the Ruby code is roughly half the length of its C# counterpart; in general, you need less lines of Ruby code to do any task, and this is confirmed in the blog "Following the rewrite" (http://rewrite.rickbradley.com/), where there is a follow-up of a rewrite of a Java application in Ruby on Rails:

> Preliminary tests by our Technical Lead put the code reduction for a normal module in our Java stack converted to Rails at roughly 20:1.

(Rick Bradley, 2005)

And whoever says less lines of code, says also less lines to read, understand and maintain in the future.

Conclusion

Each language has its advantages and disadvantages; in the case of C# and Ruby, I find both languages to be perfectly complementary, while having some very important characteristics in common. I think that both languages will be more used in the future, and that the interoperability between them will be greater as well.

References

Microsoft, "The C# Language" [Internet], http://msdn.microsoft.com/vcsharp/programming/language/ (Accessed February 26th, 2006)

Rick Bradley, "Evaluation: moving from Java to Ruby on Rails for the CenterNet rewrite" [Internet], http://rewrite.rickbradley.com/pages/moving_to_rails/ (Accessed February 26th, 2006)

Ruby web site [Internet], http://www.ruby-lang.org/ (Accessed February 26th, 2006)

TIOBE Programming Community Index [Internet], http://www.tiobe.com/tpci.htm (Accessed February 26th, 2006)

Wikipedia, “C Sharp” [Internet], http://en.wikipedia.org/wiki/C_Sharp_programming_language (Accessed February 26th, 2006)

Wikipedia, “Ruby programming language” [Internet], http://en.wikipedia.org/wiki/Ruby_programming_language (Accessed February 26th, 2006)

About OOP and Other Programming Paradigms

Adrian Kosmaczewski

2006-05-10

Does OOP reflect a “natural” way of thinking? Is it a better choice than the procedural programming paradigm?

In computer science, to say that one approach is “better” than another is to miss a great detail: I do not think that there are “better” or “natural” paradigms per se, but just appppropriate answers to certain problems in a given context.

In his 1962 book “The Structure of Nature Revolutions”, Thomas Kuhn introduces the idea of the “paradigm shift”; following this idea, human knowledge does not evolve gradually, but rather in discontinuous jumps, called “paradigm shifts” or “scientific revolutions”:

> “A scientific revolution occurs, according to Kuhn, when scientists encounter anomalies which cannot be explained by the universally accepted paradigm within which scientific progress has thereto been made. The paradigm, in Kuhn’s view, is not simply the current theory, but the entire worldview in which it exists, and all of the implications which come with it. There are anomalies for all paradigms, Kuhn maintained, that are brushed away as acceptable levels of error, or simply ignored and not dealt with (a principal argument Kuhn uses to reject Karl Popper’s model of falsifiability as the key force involved in scientific change).”

(Wikipedia, 2006)

In the case of the activity of software engineering, the paradigm shift from procedural to object-orientation is quite evident, both historically and technically speaking.

The first programming language to introduce the concept of object-oriented programming was Simula, at the beginning of the 60s; Simula (http://www.engin.umd.umich.edu/CIS/course.des/cis400/simula/simula.html, http://staff.um.edu.mt/jskl1/talk.html), as the name implies, was developed by scientist of the Norwegian Computing Center in Oslo, to simulate real-life events; thus, Simula was designed with a special objective and problem context in mind.

10 years after Simula, the Smalltalk (http://www.smalltalk.org/) programming language was created by Alan Kay in the Xerox Palo Alto Research Center (Xerox PARC, http://www.parc.com/) in Silicon Valley, to support the first computer featuring a full GUI (Graphical User Interface), the "Alto". The Smalltalk language and the Alto computer were both groundbreaking at the time (around 1973) and they paved the way to the modern personal computer as we know it today. Moreover, the object-oriented paradigm appeared as the most appropriate one to design a "windowed" system, with buttons and menus, operated by a mouse. This fact, coupled with the popularity of the C++ programming language since the 80s, set the tone for the mainstream trend in the creation of software until the appearance of Java in 1995.

Having said this, it is worth noting that the first programming toolkit for the Macintosh was designed to be used with the Pascal programming language (one of the most well-known procedural programming languages):

> "Parts of the original Macintosh operating system were written in Pascal and Motorola 68000 assembly language (though later versions incorporated substantial amounts of C++ as well), and the most frequent high-level language used for development in the early Mac community was Pascal."

(Wikipedia, 2006)

It is, then, difficult to talk about a "better" approach, when comparing OOP to procedural programming; actually, each problem must be evaluated in detail to find the paradigm that gives the best answer, not only in terms of performance and compatibility, but also in terms of maintainability.

Finally, it must be said that procedural and object-orientation are not the only programming paradigms:

- Structured programming - compared to Unstructured programming
- Imperative programming, compared to Declarative programming
- Message passing programming, compared to Imperative programming
- Procedural programming, compared to Functional programming
- Value-level programming, compared to Function-level programming
- Flow-driven programming, compared to Event-driven programming
- Scalar programming, compared to Array programming
- Class-based programming, compared to Prototype-based programming (within the context of Object-oriented programming)
- Constraint programming, compared to Logic programming
- Component-oriented programming (as in OLE)
- Aspect-oriented programming (as in AspectJ)

- Rule-based programming (as in Mathematica)
- Table-Oriented Programming (as in Microsoft FoxPro)
- Pipeline Programming (as in the UNIX command line)
- Post-object programming
- Subject-oriented programming
- Reflective programming
- Dataflow programming (as in Spreadsheets)
- Policy-based programming
- Annotative programming - http://www.flare.org

(Wikipedia, 2006)

References

Smalltalk History [Internet], http://www.smalltalk.org/smalltalk/history.html (Accessed February 26th, 2006)

Wikipedia, “Pascal programming language” [Internet], http://en.wikipedia.org/wiki/Pascal_programming_language (Accessed February 26th, 2006)

Wikipedia, “Programming paradigm” [Internet], http://en.wikipedia.org/wiki/Programming_paradigm (Accessed February 26th, 2006)

Wikipedia, “Thomas Kuhn” [Internet], http://en.wikipedia.org/wiki/Thomas_Kuhn (Accessed February 26th, 2006)

A Case for Compilers: a Fake Paper

Adrian Kosmaczewski

2006-05-13

This morning I said to myself; I should write something for my blog… but had no subject to write about; so I found a faster way: I went to http://pdos.csail.mit.edu/scigen/ and asked the SCIgen Automatic CS Paper Generator to create a document suitable to post here, to keep you entertained for a while. The results are amazing!

Enjoy! :)

Abstract

The cryptoanalysis approach to web browsers is defined not only by the development of DNS, but also by the important need for the lookaside buffer [1]. In this paper, we confirm the improvement of digital-to-analog converters, which embodies the confirmed principles of robotics. Here, we use permutable information to validate that write-back caches and kernels can cooperate to realize this intent.

Table of Contents

- 1) Introduction
- 2) Related Work
- 3) Framework
- 4) Highly-Available Modalities
- 5) Results
  - 5.1) Hardware and Software Configuration
  - 5.2) Experimental Results
- 6) Conclusion

1 Introduction

Many analysts would agree that, had it not been for robots, the visualization of hierarchical databases might never have occurred. We view cryptoanalysis as following a cycle of four phases: study, management, creation, and management. Next, Without a doubt, the usual methods for the private unification of RPCs and vacuum tubes

do not apply in this area. Clearly, perfect information and the visualization of lambda calculus do not necessarily obviate the need for the investigation of hierarchical databases.

A key solution to answer this problem is the visualization of model checking. While it at first glance seems unexpected, it has ample historical precedence. Contrarily, this method is rarely considered appropriate. Contrarily, the study of hash tables might not be the panacea that information theorists expected [2]. We emphasize that our application manages interactive epistemologies. For example, many methods measure secure algorithms. Obviously, we prove not only that agents and XML are rarely incompatible, but that the same is true for Smalltalk.

We explore a novel system for the deployment of IPv6, which we call GRIPPE. we view operating systems as following a cycle of four phases: creation, management, storage, and synthesis. Contrarily, this approach is generally adamantly opposed. Next, indeed, telephony and the Internet have a long history of connecting in this manner [3]. Combined with the producer-consumer problem, such a hypothesis enables new game-theoretic information.

Our contributions are as follows. We concentrate our efforts on disconfirming that Markov models can be made atomic, homogeneous, and probabilistic. We skip these results until future work. Similarly, we examine how fiber-optic cables can be applied to the emulation of massive multiplayer online role-playing games.

The rest of this paper is organized as follows. Primarily, we motivate the need for erasure coding. We demonstrate the visualization of online algorithms. In the end, we conclude.

2 Related Work

Several distributed and optimal heuristics have been proposed in the literature. Despite the fact that Nehru also presented this approach, we synthesized it independently and simultaneously. Our approach to DHCP differs from that of Thomas et al. as well.

While we know of no other studies on large-scale modalities, several efforts have been made to analyze cache coherence. Here, we overcame all of the obstacles inherent in the existing work. Unlike many related solutions [4], we do not attempt to deploy or learn the memory bus. Recent work by Martin et al. [5] suggests a methodology for requesting the study of hash tables, but does not offer an implementation. This work follows a long line of previous algorithms, all of which have failed [1]. Ultimately, the algorithm of Karthik Lakshminarayanan et al. is a robust choice for online algorithms.

GRIPPE builds on prior work in cooperative information and cryptography [6,7,8]. Next, Y. Jones et al. [9] developed a similar framework, contrarily we validated that GRIPPE runs in Q(2n) time. Further, a

wearable tool for exploring kernels proposed by White fails to address several key issues that our application does fix [10,11]. Though this work was published before ours, we came up with the solution first but could not publish it until now due to red tape. Obviously, despite substantial work in this area, our approach is clearly the application of choice among security experts.

3 Framework

Suppose that there exists the Turing machine such that we can easily improve the evaluation of IPv6. Though hackers worldwide largely believe the exact opposite, GRIPPE depends on this property for correct behavior. Furthermore, we believe that the well-known pervasive algorithm for the refinement of digital-to-analog converters is optimal. GRIPPE does not require such a robust synthesis to run correctly, but it doesn't hurt [3]. Figure 1 details an architectural layout depicting the relationship between GRIPPE and the location-identity split. We withhold these results due to space constraints. Figure 1 details the schematic used by GRIPPE.

Reality aside, we would like to refine a framework for how GRIPPE might behave in theory. This may or may not actually hold in reality. We show the relationship between our framework and electronic modalities in Figure 1. Furthermore, the model for our heuristic consists of four independent components: the study of hash tables, the deployment of A* search, collaborative communication, and fiber-optic cables. Figure 1 diagrams an architecture plotting the relationship between our system and metamorphic symmetries.

4 Highly-Available Modalities

GRIPPE is elegant; so, too, must be our implementation. Furthermore, the collection of shell scripts contains about 126 semi-colons of ML. GRIPPE is composed of a codebase of 25 Scheme files, a hacked operating system, and a centralized logging facility. Next, information theorists have complete control over the centralized logging facility, which of course is necessary so that spreadsheets can be made atomic, cacheable, and permutable. We have not yet implemented the hacked operating system, as this is the least essential component of GRIPPE. one cannot imagine other solutions to the implementation that would have made architecting it much simpler.

5 Results

Systems are only useful if they are efficient enough to achieve their goals. We did not take any shortcuts here. Our overall performance analysis seeks to prove three hypotheses: (1) that erasure coding no longer affects median interrupt rate; (2) that symmetric encryption have actually shown muted mean response time over time; and finally

(3) that the Commodore 64 of yesteryear actually exhibits better average sampling rate than today's hardware. Our logic follows a new model: performance might cause us to lose sleep only as long as usability constraints take a back seat to performance. We hope that this section illuminates D. B. Li's analysis of the memory bus that would allow for further study into Byzantine fault tolerance in 1967.

5.1 Hardware and Software Configuration

A well-tuned network setup holds the key to an useful evaluation method. We executed a simulation on our system to measure ubiquitous modalities's impact on the work of Canadian mad scientist V. Kumar. This step flies in the face of conventional wisdom, but is essential to our results. To begin with, we removed 200 FPUs from our mobile telephones to measure the collectively peer-to-peer behavior of random symmetries. This configuration step was time-consuming but worth it in the end. Further, we removed 7Gb/s of Ethernet access from our decommissioned Motorola bag telephones to measure opportunistically "smart" configurations's effect on the work of American gifted hacker Stephen Hawking. We added 7MB of NV-RAM to our millenium cluster. This configuration step was time-consuming but worth it in the end. Next, we quadrupled the effective floppy disk speed of the KGB's network to discover the effective hard disk space of MIT's sensor-net testbed. Finally, we tripled the effective NV-RAM throughput of our desktop machines to better understand our stable testbed.

GRIPPE does not run on a commodity operating system but instead requires a collectively exokernelized version of Microsoft Windows NT. we implemented our Moore's Law server in embedded C, augmented with randomly random extensions [13]. Our experiments soon proved that microkernelizing our tulip cards was more effective than distributing them, as previous work suggested. Further, this concludes our discussion of software modifications.

5.2 Experimental Results

Our hardware and software modficiations show that simulating GRIPPE is one thing, but deploying it in a controlled environment is a completely different story. That being said, we ran four novel experiments: (1) we measured tape drive space as a function of hard disk speed on a PDP 11; (2) we deployed 13 IBM PC Juniors across the planetary-scale network, and tested our link-level acknowledgements accordingly; (3) we dogfooded our framework on our own desktop machines, paying particular attention to effective ROM speed; and (4) we ran 35 trials with a simulated DHCP workload, and compared results to our software emulation. All of these experiments completed without noticable performance bottlenecks or access-link congestion.

Now for the climactic analysis of the second half of our experiments. Error bars have been elided, since most of our data points fell outside of 43 standard deviations from observed means. Further, the data in Figure 3, in particular, proves that four years of hard work were wasted on this project. Continuing with this rationale, the many discontinuities in the graphs point to duplicated interrupt rate introduced with our hardware upgrades.

We have seen one type of behavior in Figures 4 and 2; our other experiments (shown in Figure 2) paint a different picture. The key to Figure 4 is closing the feedback loop; Figure 2 shows how our algorithm's tape drive space does not converge otherwise. Similarly, operator error alone cannot account for these results. The key to Figure 2 is closing the feedback loop; Figure 2 shows how our heuristic's RAM throughput does not converge otherwise.

Lastly, we discuss experiments (3) and (4) enumerated above. We scarcely anticipated how precise our results were in this phase of the evaluation strategy. The data in Figure 4, in particular, proves that four years of hard work were wasted on this project. Such a claim at first glance seems unexpected but has ample historical precedence. Gaussian electromagnetic disturbances in our 10-node testbed caused unstable experimental results.

6 Conclusion

In conclusion, in this paper we validated that the famous homogeneous algorithm for the deployment of kernels by Davis and Robinson [15] is in Co-NP. Continuing with this rationale, we also motivated a secure tool for improving Boolean logic. Continuing with this rationale, we also motivated a heuristic for encrypted configurations. We also introduced an analysis of operating systems.

References

[1]

C. Leiserson, "An analysis of the memory bus using ASHES," in POT WMSCI, July 1998.

[2]

L. Subramanian, J. Wilkinson, E. Harris, A. Kosmaczewski, J. Garcia, C. Leiserson, and C. A. R. Hoare, "An improvement of compilers using Prinker," in POT the Conference on Atomic, Mobile Information, Sept. 1993.

[3]

E. Dijkstra, H. B. Moore, C. Darwin, and W. Kahan, "Exploration of the lookaside buffer that paved the way for the development of extreme programming," OSR, vol. 27, pp. 59-68, Sept. 1993.

[4]

S. Watanabe and C. Johnson, “Decoupling superblocks from link-level acknowledgements in suffix trees,” NTT Technical Review, vol. 4, pp. 78-87, Apr. 2000.

[5]

D. Bharath, “Improving the UNIVAC computer and vacuum tubes,” in POT the Workshop on Metamorphic Theory, Oct. 1997.

[6]

C. Leiserson and J. Hennessy, “Deploying Markov models and IPv4 using Jugulum,” in POT the Workshop on Semantic Models, Jan. 1990.

[7]

J. Zheng, B. Lampson, and D. Patterson, “On the visualization of symmetric encryption,” in POT HPCA, Apr. 2002.

[8]

K. Nygaard, “Decoupling gigabit switches from erasure coding in symmetric encryption,” in POT the Conference on Authenticated Theory, July 1995.

[9]

H. Levy, R. Qian, and S. Bose, “Developing kernels and robots,” in POT the Workshop on Collaborative, Client-Server Modalities, Aug. 2001.

[10]

R. Milner, U. Gupta, F. L. White, and L. Subramanian, “Roband: A methodology for the deployment of cache coherence,” in POT the Workshop on Authenticated, Psychoacoustic Epistemologies, July 1994.

[11]

O. P. Ito, “Contrasting sensor networks and neural networks using ENDICT,” in POT HPCA, Mar. 1986.

[12]

A. Newell and K. Thompson, “Deconstructing evolutionary programming with Doole,” in POT OOPSLA, May 1993.

[13]

A. Kosmaczewski, “A case for suffix trees,” in POT MOBICOM, July 2005.

[14]

B. Lampson and D. Ritchie, “PILON: A methodology for the study of spreadsheets,” in POT SIGGRAPH, Mar. 2004.

[15]

R. W. Zhao and M. Minsky, “Self-learning, low-energy information for context-free grammar,” in POT the Conference on Stable, Lossless Models, July 2003.

How to Install the Linksys WPC54GS Wireless G Network Adapter in Ubuntu 5.10 Breezy

Adrian Kosmaczewski

2006-05-19

Well, that was a long title.

My mother recently changed her 4-year old Dell Inspiron 4100 for a brand new iMac G5, so I took that old laptop (20 GB hard disk, 256 MB RAM) and, after backing up her data, wiped it completely and reinstalled it with the following configuration:

- Windows 2000 Professional SP4 (I just hate Windows XP, and think that Windows 2000 is the best ever Windows)
- Ubuntu[1] 5.10 “Breezy”

Then I bought a Linksys Wireless-G Notebook Adapter with SpeedBooster[2], to be able to use my wireless connection at home (in 2002 Dell notebooks did not have wireless LAN adapters by default). Of course, installing it in Windows 2000 was easy, but the difficult part was to have it running in Ubuntu...

Of course I could have taken a look in the web before buying it, but hey, I love living dangerously :)

I am amazed to see that the Ubuntu community keeps trying anything on it, so that I quickly found lots of information about how to make the damn card work. But sometimes that information can be quite misleading.

First of all I found this page in the Ubuntu wiki[3], that simply says that the WPC54GS is 100% compatible and states “Plug it in, reboot, you’re done”. Sorry to say this guys, but at least in my Ubuntu installation, this did not work.

Then I read somewhere about ndiswrapper[4], which allows to use native wireless Windows XP drivers directly in any Linux flavor. Yep, the

[1] http://www.ubuntu.com

[2] http://www1.linksys.com/products/product.asp?prid=611&scid=36

[3] https://wiki.ubuntu.com/HardwareSupportComponentsWirelessNetworkCards

[4] http://ndiswrapper.sourceforge.net/

same drivers with those ".inf" files that you use in Windows, you can use them in your Linux installation. Awesome, what can I say.

So this is how I came up with this page about Ubuntu in the ndiswrapper MediaWiki[5] where everything is explained; this page points to another one[6], where the basic installation procedures are explained in great detail.

This is what I did:

1. I installed `ndiswrapper-utils` using the System -> Administration -> Synaptic Package Manager
2. I inserted the CD that came with the card in the laptop
3. In a terminal window, typed `sudo ndiswrapper -i /cdrom/lsbcmnds.inf` (you must be root)
4. If everything goes OK, if you type `ndiswrapper -l` you should see the newly installed driver
5. Then I typed `sudo ndiswrapper -m` to make this driver load every time the machine boots up (that's what I understood that the thing did... maybe I'm wrong!)
6. Then I typed `sudo depmod -a`; if no errors appear, then everything is OK
7. Then I typed `sudo modprobe ndiswrapper` which loads the module in memory, right now.
8. After that, I opened System -> Administration -> Networking, and found (phew!) that the Wireless network adapter was finally recognized by Ubuntu. I configured it the same way as in Windows (the ESSID of the network, the security type, the credentials), and voilà, my Ubuntu laptop is online!

I had already installed Ubuntu in my G3 iBook (see my previous post[7]) but I must say that I prefer it in this Inspiron laptop rathen than in my iBook. It just feels better, and faster.

Anyway, hope this helps!

Update, 2006-05-01: I have made the card work in Kubuntu as well, check it out![8] It is slightly more complicated...

Also, thanks to whoever put a link to this page from https://wiki.ubuntu.com/HardwareSupportComponentsWirelessNetworkCards :)

Update, 2006-06-02: I have made the card work in Ubuntu Dapper 6.06 as well, check it out![9] The trick is to blacklist the default Broadcom driver... ;)

---

[5] http://ndiswrapper.sourceforge.net/mediawiki/index.php/Ubuntu

[6] http://ndiswrapper.sourceforge.net/mediawiki/index.php/Installation#Install_Windows_driver

[7] /blog/ubuntu/

[8] /blog/kubuntu-5.10-and-the-linksys-wpc54gs-wireless-g-network-adapter/

[9] /blog/how-to-install-the-linksys-wpc54gs-wireless-g-network-adapter-in-ubuntu-6.06-dapper/

Intelligent Software Agents - a .NET Example

Adrian Kosmaczewski

2006-05-25

Software Example

In the February 2006 issue of MSDN Magazine (http://msdn.microsoft.com/msdnmag/), Matt Neely describes a .NET implementation of mobile agents:

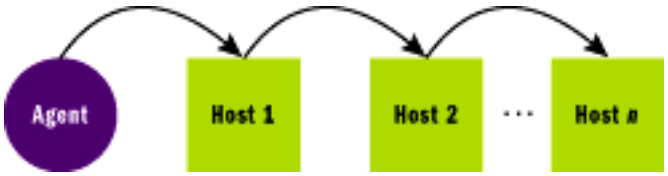
> The term agent originates in artificial intelligence and describes a logical entity that has some level of autonomy within its environment or host. A mobile agent has the added capability to move between hosts. In a computing context, a mobile agent is a combined unit of data and code that can move between different execution environments.

(Neely, 2006)

The idea described in the article is that of a small family of .NET classes that literally "jump" from a computer to another, performing tasks in the host computer, through a mechanism called Remoting:

> An example of a traveling agent app could perform operations control. An agent is sent out with a list of machines on the local network it should traverse to inventory hardware and software (…)
>
> built-in services that facilitate the componentization and mobility of code, namely object remoting and serialization.(…)
>
> Mobile agents have their uses and their pros and cons. The autonomous and mobile nature of mobile agents can lead to reduced network traffic, decentralization, increased robustness and fault-tolerance, and easy deployment.

(Neely, 2006)



(Source: Neely, 2006)

Such an agent could be used for administrative purposes (for example, a system administrator could use it to perform complex tasks directly in the target host computer), for inventory purposes (using WMI, such an agent could "collect" information about the computer), and even, and most dangerously, to inject viruses or trojans (such a mechanism could open security backdoors to external attacks in a computer!).

The author goes on describing three archetypical use cases for such agents:

- A simple travelling agent ("Travelling Pattern");
- A task agent ("Task Pattern", handling distributed workload in a network, similar to the approach used by the SETI@home project);
- A buyer/seller scenario ("Interactive Pattern") with a buyer that looks for the best price in several sellers running in different computers (this example shows the biggest complexity).

For more information about .NET Remoting, please refer to this article: http://msdn.microsoft.com/library/en-us/dndotnet/html/hawkremoting.asp. For the moment, suffice to say that Remoting allows to instantiate and execute objects and methods in computers connected by any communication medium, in a protocol-independent way (that is, using HTTP, TCP, etc). For instance, the interaction with remote SOAP web services can be seen as a particular type of Remoting.

Comparison

In the case of the hardware agent described by Brookshear, he describes the following characteristics as those defining an intelligent agent:

- Rational reaction to stimuli
- Intelligent behavior
- Learning capabilities

Let's see how the software example of the MSDN Magazine article compares to a hardware agent in these three specific fields.

Rational Reaction to Stimuli

> An agent is a "device" that responds to stimuli from its environment. Most agents have sensors by which they receive data from their environments. Example of sensors such include microphones, cameras, and air sampling devices.

(Brookshear, 2005)

In the case of the software agent described in the article, the "sensors" are the complete API exposed by the .NET Framework. The agent can use any of the methods exposed in it to get information about the host where the agent is being executed. For example, in the case of the

task agent (second example), the agent gets the list of logical drives in the current host computer:

> My child agent will merely list the system's logical drives (with a call to Directory.GetLogicalDrives) and again output the results to the host's console.

(Neely, 2006)

Intelligent Behavior

> The goal of artificial intelligence is to build agents that behave intelligently. This means that the actions of the agent's actuators must be rational responses to the data received through its sensors.

(Brookshear, 2005)

In the article, the author describes the "Interaction Pattern" of a buyer/seller scenario:

> MyBuyingAgent (…) is a bit more complex as it needs to maintain more state. The buying agent is given the name of an item to buy, a value for the maximum amount it can spend for that item, and an array of host URLs that the agent needs to visit in order to find the item at the lowest price.
>
> As the buying agent travels, it will need to enumerate each seller on each host, determine if that seller has the item desired, and then find the item's price. Since the agent wants to find the best price, it will have to track which seller it is going to buy from. I call this the winning seller. The agent will have to store the location of the winning seller, the ID of the winning seller, and the price the winning seller is offering for the desired item.

(Neely, 2006)

Here we have clearly a description of an intelligent behavior of the buyer agent, following the description given by Brookshear.

Learning Capabilities

> In some cases an agent's responses may improve over time as the agent learns. This could take the form of developing procedural knowledge (learning "how") or storing declarative knowledge (learning "what").

(Brookshear, 2005)

This last characteristic is not explicitly mentioned in the article, and with a good reason; it is by far the most complex of all, and would require a more complex design. The author of the MSDN Magazine article has purposedly kept the article simple, to introduce the major concepts and the general idea.

Conclusion

The article in MSDN Magazine does not delve into the learning mechanisms, that would make this “agent” a complete one in the AI sense. It rather focuses more into communication and security issues, giving at the same time the pros and cons of such an implementation.

My conclusion is that, given the proper learning instructions, the software agent would be fully compliant with the requirements set by the AI discipline.

References

J. Glenn Brookshear, “Computer Science, An Overview, Eighth Edition”, ISBN 0-321-26971-3, Addison Wesley, 2005

Matt Neely, “Write Mobile Agents In .NET To Roam And Interact On Your Network”, MSDN Magazine, February 2006 (Available here: http://msdn.microsoft.com/msdnmag/issues/06/02/MobileAgents/default.aspx - Accessed March 18th, 2006)

How to Install the Linksys WPC54GS Wireless-G Network Adapter in Ubuntu 6.06 Dapper

Adrian Kosmaczewski

2006-06-02

"Dapper", the new version of Ubuntu has been released yesterday! So I downloaded the ISO file, burn the CD down and proceeded to install it over my old Breezy Kubuntu installation.

Of course, I wanted to use my good' old Linksys WPC54GS Wireless-G Network Adapter with Broadcom chipset, and I was lucky enough to find this page:

https://wiki.ubuntu.com/WifiDocs/Driver/bcm43xx?action=show&redirect=WifiDocs%2FDriver%2FBroadcom43xx#head-e70dd6b5c57894d32e3eddc4f3e21d7d6d02230f

It describes the whole Broadcom problem in Ubuntu, and gives the instructions needed to make the card work in Dapper:

1. Follow the instructions in this blog post[1] from steps 1 to 7 (step 8 cannot be done, yet)
2. Type "echo 'blacklist bcm43xx' | sudo tee -a /etc/modprobe.d/blacklist"
3. Type "sudo rmmod bcm43xx", "sudo rmmod ndiswrapper" and "sudo modprobe ndiswrapper"
4. Do the step 8 of this post[2]

And that's it! You've got the wireless card ready for use.

:) I'm happy to have found that page! It seems that the Broadcom drivers that come with Dapper do not work with all concerned wireless cards, so the thing is to "blacklist" them, and then to load ndiswrapper that works perfectly well.

[1] /blog/how-to-install-the-linksys-wpc54gs-wireless-g-network-adapter-in-ubuntu-5.10-breezy/

[2] /blog/how-to-install-the-linksys-wpc54gs-wireless-g-network-adapter-in-ubuntu-5.10-breezy/

How to Install Ubuntu 6.06 Dapper in an Apple G3 iBook

Adrian Kosmaczewski

2006-06-03

After upgrading my good ol' PC[1] to Dapper, I proceeded to download and burn a copy of it for PowerPC systems, and I have managed to install it in my old faithful G3 iBook in dual boot, with Mac OS X 10.2 "Jaguar" on the other partition. Here's how I did it.

1. **Backup all the information in your computer; what we are about to do erases your hard disk completely!!! You have been warned!**
2. Insert the Jaguar CD in the drive and reboot; **to boot from the CD, remember to press the "C" key on your keyboard when you hear the chime of the iBook booting**
3. In the Mac OS X installer, go to the "Install" menu and select "Disk Utility"
4. In the Disk Utility, partition the hard drive in two parts with the same size; there is a special drop-down menu in the "Partition" tab that provides this automatically
5. Select the first partition, the one at the top, and choose "Free space" from the "Format" drop down menu at the right.
6. Select the second partition, at the bottom, and choose "Mac OS Extended" from the "Format" drop down menu.
7. Quit Disk Utility, and Install Mac OS X as usual
8. When Jaguar is installed, insert the Ubuntu disk and reboot; **to boot from the CD, remember to press the "C" key on your keyboard when you hear the chime of the iBook booting** (I prefer to repeat it :)
9. Launch the installation of Ubuntu Dapper, and when you are asked about the partition where to install it, select the second option: "Use the largest block of continuous free space on disk"
10. Install Ubuntu as usual after that

The good news is that Ubuntu Dapper recognized everything in the iBook G3; even the Airport wireless card! I was afraid that it wouldn't... but it worked perfectly well. I joined my home network without any problem at all.

[1] /blog/how-to-install-the-linksys-wpc54gs-wireless-g-network-adapter-in-ubuntu-6.06-dapper/

And even better: I can put my computer to sleep, and wake it afterwards without any problem at all! I was asked by Ubuntu to “enable” the sleep functionality (in Breezy it did not work at all, but here it works perfectly well!).

Only funny thing: to simulate a right-click, I have to press the “F12” key :)))

Ubuntu Dapper is awesome :)

Starting Again

Adrian Kosmaczewski

2006-09-26

So here we go again. They say that the only constant thing is change.

Back to blogging, but with a few differences. Not only on the screen, but in real life too.

This time, I drop the idea of maintaining several blogs at once, as I did before with those previous blogs. It was much too work :-P actually, only two of those were purely mine, since "Stop the Press" was more a blackboard to put interesting stuff that I find elsewhere, trying to keep it in a handy place. As a matter of fact I always feared that someone e-mailed me telling me to take those copies offline, but it never happened, and I hope it doesn't; I like to have all of that online.

I also drop Movable Type[1] altogether, and instead choose Wordpress[2]. For several reasons:

- Wordpress is open source software[3]; Movable Type is not, although it is free;
- Wordpress is easy to install, to use, etc, etc, etc.;
- Wordpress has lots of cool plugins[4] and themes[5];
- Wordpress has a rich text editor for posts that works also on the Mac;
- Wordpress looks and feels nicer (purely subjective)
- Scobleizer[6] uses Wordpress :D

The old blogs will remain at the same URL as before, since some posts (particularly those related to Ubuntu and the Linksys wireless card configuration) are referenced from other sites, and I do not want people to get lost. Those pages are very important to me, and hopefully to others as well.

And in my life... well, Claudia[7] came to Switzerland and we're finally married[8]. Life starts again, **better than ever.** My working life has

[1] http://www.movabletype.org/
[2] http://www.wordpress.org
[3] http://www.opensource.org/
[4] http://wp-plugins.net/
[5] http://themes.wordpress.net/
[6] http://scobleizer.wordpress.com/
[7] time-2005-04-17.pdf
[8] /wedding/

also changed substantially; I no longer work with or for Microsoft technologies; I am now senior C++ software developer[9] in riskpro technologies AG[10], happily using and discovering wxWidgets[11], Boost[12], Ubuntu[13], Ruby on Rails[14] and lots of other cool things going around, free to discover, to enjoy, to use, and to fix, if needed. I have discovered lots of cool stuff, and everyday the open source world keeps me more amazed, entertained, and productive than ever before; this is something that Microsoft ceased to do long ago…

So here we go again, because I still have a lot to say. Stay tuned!

[9] http://www.iris.ch/en/career2.asp?REF_ID=riskpro%20senior%20developer
[10] http://www.rptec.ch/
[11] http://www.wxwidgets.org/
[12] http://www.boost.org/
[13] http://www.ubuntu.com/
[14] http://www.rubyonrails.org/

Google Code Search vs. Koders.com

Adrian Kosmaczewski

2006-10-05

Last year I've written[1] about Koders.com[2], a search engine that crawled open source code repositories and allowed developers to search for code; an extremely interesting and valuable tool indeed. When I first saw Koders.com I thought (and I wrote that down as well) that they would be soon bought by Google, because everything in Koders.com looks at first glance like a Google application. But actually, something different happened: Google came up with Google Code Search[3], its own code search engine. Here's a quick comparison of both.

The first difference between both systems is the interface; Google Code Search, as the other Google applications, has a simpler interface and uses a "command-line syntax" to narrow searches: for example, if you type "lang:c++"#include " it will bring all files containing GPL code written in C++ that include the STL stack template class[4]. Trying the same search in Koders.com means to select the proper values in the drop-down menus, and you get somewhat different results[5]. Then you can click on any filename of the result list, and see the code inside. No need to browse SourceForge[6] or Tigris[7]; fast and easy. Very neat.

I must say that I like the command-line thingy of Google (I just use it every day in the "normal" Google, doing things like "filetype:pdf" to narrow searches) and I found that Google Code Search brings more relevant results (heh, this is Google after all). But... the code preview of Koders.com is still above that of Google. Koders.com provides nicer syntax highlighting for every possible language, and the special pages for projects (for example, this is the page of the Subversion project[8]) are simply out of this planet. On the other hand, Google al-

[1] /blog/new-tools-for-open-source-developers-and-others-as-well/

[2] http://www.koders.com/

[3] http://www.google.com/codesearch

[4] http://www.google.com/codesearch?hl=en&lr=&q=license%3Agpl+lang%3Ac%2B%2B+%22%23include+%3Cstack%3E%22&btnG=Search

[5] http://www.koders.com/?s=%22%23include+%3Cstack%3E%22&_%3Abtn=Search&_%3Ala=Cpp&_%3Ali=GPL

[6] http://sourceforge.net/

[7] http://www.tigris.org/

[8] http://www.koders.com/info.aspx?c=ProjectInfo&pid=ZC9RRQ78P713PB1DWQRVRMNBNB

lows you to search the contents of tar.gz or zip files, when they match your search criteria.

If I can make a wish, this is it: Google buys Koders.com, they merge the display technology of Koders.com and the search capabilities of Google into a simple product. Now that would be great. But... maybe both tools are targeting different but complementary needs? In any case, they are both welcome in my toolkit.

PS: I learnt about Google Code Search after reading this posting[9] from Scott's Theocacao[10]. Thanks!

Update, October 13th, 2006: You can really find anything[11] in Koders and Google Code Search :)

[9] http://www.theocacao.com/document.page/312
[10] http://www.theocacao.com/
[11] http://www.regdeveloper.co.uk/2006/10/13/code_outrage/

Avoiding Basic Trouble

Adrian Kosmaczewski

2006-10-06

I remember that, late 2004, I was asked by my employer to evaluate the migration of a huge (huge, did I say huge?) Visual Basic 6 "classic" client-server application to an SOA-based Visual Basic .NET one. The application was a business-critical one for several customers, kind of a government ERP system, built initially in VB 3 or 4, and slowly migrated through the years to new versions of VB. Until VB.NET came out.

Now for those that might not know it, even if those two versions of Visual Basic come from the same company, well, they are not[1] AT[2] ALL[3] compatible. That's how trouble came in.

The problem is that VB6 and VB.NET are architecturally different. They have huge differences in terms of underlying mechanisms, data types and internal protocols, and you just cannot migrate a VB6 app "the easy way", doing "File/Import/VB6 Project". It is just not possible at all. Worst of all, these technical words, about architecture and so on, you cannot tell them to your clients; they just say "but it's Visual Basic, right? So why don't you just port it and shut up?". They are blinded by Microsoft's all mighty presence. The consultant has to suffer. That's how they see things.

The migration was even more difficult because the application was worth 500 KSLOC, had not the slightest separation between tiers (yes, it was good ol' plain vanilla VB6, with UI code mixed with business and persistence code), it has not been documented at all, and the only way to know the internals of the thing was to ask the lead developer… who was all fed up with the management of our company, and was about to leave. So you can imagine the scenario.

I should have proposed to use RealBASIC[4]. These are smart people; they have figured out that there is a huge quantity of people[5] still operating with "legacy" (ahem) VB6 code, and that they would happily use .NET if they only could. So RealBASIC came up with a product that:

[1] http://www.thescarms.com/vbasic/VB6vsVBNet.asp
[2] http://en.wikipedia.org/wiki/Visual_Basic_.NET#Controversy
[3] http://www.aivosto.com/vbtips/vbnetmigration.html
[4] http://www.realbasic.com/
[5] http://classicvb.org/petition/

- Lets you import existing VB6 code
- Provides a Basic dialect with full OOP features (polymorphism, inheritance, etc)
- Lets you build from the same source code, binaries for Linux, Windows and Mac OS (9 and X, PowerPC or Intel).

So that's what I call smart. Why hasn't Microsoft came up with something like this? Instead, in all of its arrogance[6], Microsoft has dropped millions of "classic" VB users into darkness and oblivion. I know quite a few of them; they like the fast time-to-market that you get with VB. But they are fed up with the company behind.

Now, let's imagine for a minute that VB was open-source technology:

- There would have been a strong opposition to the VB developers to drop support for earlier version; this would have resulted in a "fork" of the open-source project.
- My former employer would have surely had access to the forked Visual Basic compiler, adding true OOP features to the language without dropping support (maybe with a syntax similar to that of RealBASIC)
- The VB.NET version would have appeared, nevertheless, and new VB development would be done in that language instead.

Not only that, but there would be VB plugins for Eclipse, there would have been cross-platform compatibility for a long time, more innovative extensions here and there, and the language would not be that crappy after all. Yes, I have worked with it; some big (big, did I say big?) industrial groups have built their entire IT infrastructure with it. And they are paying the price of using a non-standard, closed, proprietary programming language, with increased costs and migration problems (now and tomorrow).

Now, if you ask me, my strategy, if I were to create a cross-platform application, would be the following:

- Use standard ISO C++ for the backend;
- Use SQLite[7] for storing info in files (instead of pure binary files);
- Use wxWidgets[8] for the frontend (Win and Linux);
- Use Cocoa[9] and Objective-C++[10] for the Mac frontend.

Then, have everything stored in a Subversion[11] repository, use Eclipse[12] with CDT[13] as IDE, document it with Doxygen[14], and use

[6] http://www.businessweek.com/1999/99_11/b3620035.htm
[7] http://www.sqlite.org/
[8] http://www.wxwidgets.org/
[9] http://developer.apple.com/cocoa/
[10] http://developer.apple.com/documentation/Cocoa/Conceptual/ObjectiveC/Articles/chapter_4_section_10.html
[11] http://subversion.tigris.org/
[12] http://www.eclipse.org/
[13] http://www.eclipse.org/cdt/
[14] http://www.stack.nl/~dimitri/doxygen/

CppUnit[15] for unit tests. That's how I would do it. Even if the RealBASIC approach seems interesting, I'd rather use open source technologies instead.

[15] http://cppunit.sourceforge.net/

About Cross Platform Unit Testing

Adrian Kosmaczewski

2006-10-09

A lot has been said about unit testing à la JUnit in C++; the most interesting article about the subject is without any doubt Exploring the C++ Unit Testing Framework Jungle, By Noel Llopis[1], with a thorough comparison of the most important C++ unit testing frameworks out there:

- CppUnit[2]
- Boost.Test[3]
- CppUnitLite[4]
- NanoCppUnit
- Unit++[5]
- CxxTest[6]

While I won't go into much detail about his article, I think that the only aspect that Noel forgot in his analysis is the **cross-platform capabilities** of these frameworks. And that is precisely the aspect that I was looking for while choosing a unit testing framework for a personal project: I needed it to work seamlessly in Ubuntu, Mac OS X and Windows.

I haven't looked at all of them, but I took a serious look to Boost.Test and CppUnit. The first one seems promising but much too complicated to install (you have to build the whole Boost libraries to get it - please correct me if I did wrong!), while CppUnit, that I used in my job, was a pleasure to work with. I have managed to build it directly from the sources doing a classic set of commands:

```
$ tar xvfpz cppunit-1.12.0.tar.gz
$ cd cppunit-1.12.0
$ ./configure
$ cd src/cppunit
$ make
$ sudo make install
```

[1] http://www.gamesfromwithin.com/articles/0412/000061.html%22
[2] http://cppunit.sourceforge.net/
[3] http://boost.org/libs/test/doc/index.html
[4] http://c2.com/cgi/wiki?CppUnitLite
[5] http://unitpp.sourceforge.net/
[6] http://cxxtest.sourceforge.net/

This worked perfectly well in both Mac OS X 10.2 “Jaguar”, Mac OS X 10.4 “Tiger”, and Ubuntu Dapper. It installs the library in /usr/local/lib and after that, you can use it normally as any other lib.

On Windows, I found the instructions to make it work out-of-the-box in Visual Studio .NET 2003[7], and also using Eclipse CDT + MinGW[8].

Exactly what I wanted! Cross-platform, open-source, free, with good documentation, mature code base, easy to build, easy to include in projects (in my case, using the shared object library / dylib / DLL) and easy to link to.

PS: reading the comments in Noel’s article[9] I found these references to two more C++ unit test frameworks:

- jUtAsserter[10]
- TUT[11]

[7] http://cppunit.sourceforge.net/cppunit-wiki/BuildingCppUnit1#head-e3e3499827fdb0150cf42e6f5f5d83b8cf0750ac

[8] http://cppunit.sourceforge.net/cppunit-wiki/CppUnitWithEclipse

[9] http://www.convexhull.com/mt/mt-comments.cgi?entry_id=61

[10] http://arrizza.com/unittesters/jutasserter/jutasserter.html

[11] http://tut-framework.sourceforge.net/

Xubuntu

Adrian Kosmaczewski

2006-10-12

Since I discovered Ubuntu[1] I've been trying to install it in different hardware, in different computers, even in virtual machines, and I just love it. It installs without problems, I can add and remove the coolest productivity and development tools fast and easy, everything is ready to use, and it just feels great.

My latest discovery is Xubuntu[2]. It is basically the same as Ubuntu and Kubuntu[3], but with the Xfce[4] desktop instead or GNOME[5] or KDE[6]. The nice thing is that I have not had to uninstall Ubuntu: I just typed "sudo apt-get install xubuntu-desktop" at the command prompt, and 10 minutes later I logged into Xubuntu.

The net result is a soooooooo much faster user experience.

The main machine where I have Xubuntu installed is a rather old G3 iBook that I bought in 2002, with only 256 MB of RAM and 30 GB of hard disk. The machine runs like a charm, but of course with GNOME and KDE there's a lot of swapping. As soon as I installed the Xfce desktop, things went really faster. I can only recommend using Xubuntu in old machines: the base system only takes 90 MB of RAM, and applications load faster than in GNOME or KDE.

I just love (X)(K)(Ed)Ubuntu :)

[1] http://www.ubuntu.com/
[2] http://www.xubuntu.org/
[3] http://www.kubuntu.org/
[4] http://www.xfce.org/
[5] http://www.gnome.org/
[6] http://www.kde.org/

Posting Remotely… From Google Docs and Spreadsheets!

Adrian Kosmaczewski

2006-10-12

Posting from Google Docs & Spreadsheets …

This is a test posting from what used to be Writely, the online word processing tool; it turns out that you can use it to publish in your blog, just using the API that is exposed by Wordpress!

Calculator Project

Adrian Kosmaczewski

2006-10-13

I have just posted a page in the site, about a recent project I've done, implementing a simple stack-based calculator, similar to those HP ones: it is called Calculator, and you can find it under the Project pages[1].

There are several ideas behind this project:

- Create a command-line application using C++, in such a way that the same source code could be used to create executables for the Mac OS X, Linux and Windows.
- Have a good suite of unit tests for the different modules of the application, using CppUnit[2].
- Give a test drive to Doxygen[3] and "dot[4]", to generate a full CHM file plenty of diagrams and explanations about the solution.
- Create the application using as many IDEs as possible:
  - Eclipse[5]
  - Visual Studio[6] (6.0, .NET 2003 and Express 2005)
  - Apple Project Builder
  - Apple Xcode[7]
- Compile the application using the biggest possible number of compilers:
  - Borland's free command line tools[8]
  - Microsoft (3 different versions)
  - GCC[9] (under Windows, Mac OS X and Linux)
- Learn the basic syntax of makefiles[10] and provide one for each target platform (the Windows version uses MinGW)
- Make the source code available as "Public Domain[11]" for others to test it, and play with it.

[1] calculator.zip
[2] /blog/about-cross-platform-unit-testing/
[3] http://www.stack.nl/~dimitri/doxygen/
[4] http://www.graphviz.org/
[5] http://www.eclipse.org/
[6] http://en.wikipedia.org/wiki/Microsoft_Visual_Studio
[7] http://www.apple.com/macosx/features/xcode/
[8] http://www.borland.com/downloads/download_cbuilder.html
[9] http://gcc.gnu.org/
[10] http://www.gnu.org/software/make/
[11] http://creativecommons.org/licenses/publicdomain/

I have learnt several things doing this:

- For some reason, dynamic libraries on the Mac have the “dylib” extension, while in Linux they are “so” files (“shared objects”). They are the equivalent of DLLs in Windows. Other than the naming different, they are used almost the same way.
- Eclipse is a great thing; I could open the project without problem in the three environments.
- Microsoft compilers generate much, much smaller executables than GCC; I would really know why!
- CppUnit works like a charm in all possible environments, exactly as I wanted[12].
- C++ is an incredible language. I just use 0.1% of what it can do, but I really liked it.
- Doxygen is something out of this planet.

Feel free to download it[13] and tell me your thoughts about it!

[12] /blog/about-cross-platform-unit-testing/
[13] calculator.zip

Mention on Matt Gemmell's Blog

Adrian Kosmaczewski

2006-10-28

Published by Matt Gemmell[1].

> Hmm. It's taken 4 years for me to realise that iCal Birthday Shifter shares an acronym with Irritable Bowel Syndrome. In any case, this minor update is for the former, fixing a crasher with nil-titled calendars (usually Groups; thanks to Adrian Kosmaczewski for the heads-up) and now ordering the calendars the same way as iCal instead of just alphabetically.

[1] http://mattgemmell.com/2006/10/29/software-birthdays

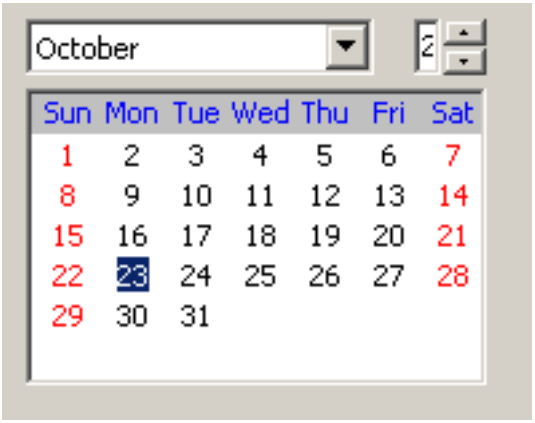
The Power of Open Source

Adrian Kosmaczewski

2006-10-31

I'm just giving the finishing touches to my wxWidgets + SQLite + CppUnit personal project, all written in standard C++, and compiling in Mac, Windows and Linux as well. And I had the opportunity to see how good is to be able to change the source code of a library when you need it.

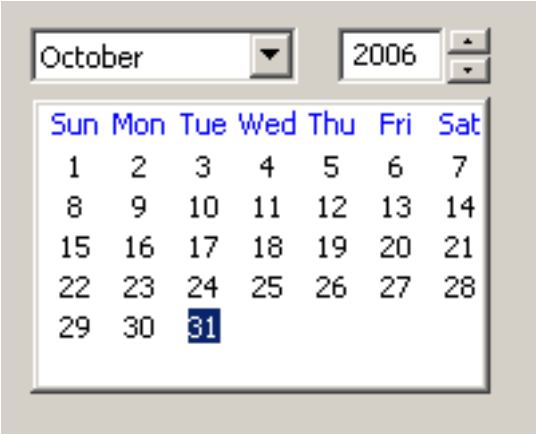
The problem was that the wxCalendarCtrl control that comes with wxWidgets 2.6.3 (see note below) did not render properly on Windows. In all the other platforms it looked OK, but in Windows it just looked like this:



As you can see, the wxSpinCtrl used to change the years is next to unusable. I looked around the API documentation to see if there was some method to change that width, but it turns out that there is not such thing.

But... since wxWidgets is open-source, I just opened the `src\generic\calctrl.cpp` file and looked inside to see how the whole thing was built: I found that the size of the spinner depends on that of the months wxComboBox control, and then what I did is just to force the size of the combo box to a fixed size of 100 pixels, instead of the default value "wxDefaultCoord" (I did the change in line 275, in the constructor of the wxComboBox called "m_comboMonth").

I rebuilt the library, and then I rebuilt my own project linking to the patched code:



Voilà! No need to dig into proprietary support website that seldom bring an answer, or to wait for a service pack or patch or anything else that will correct the thing. With open source software, I can just go and fix things myself, whenever I need it. And for free.

Note: By the way, I'm using version 2.6.3 since I want the code to compile under Mac OS X 10.2 "Jaguar" as well (as far as I saw, 2.7 is not compatible with Jaguar any more). I will post more screenshots, as well as the whole source code, very soon!

Do-It-Yourself, Now and Then

Adrian Kosmaczewski

2006-11-02

Beginning 1998, nearly 9 years ago, I created this tool at my former employer's site. It allowed customers to create their own web page, for a small fee, using all the interaction available at the time of Netscape Communicator and Internet Explorer 4. You could type some text here and there, choose some (awful, really) colors, and publish your page in a couple of minutes. A good number of people used it until 2001, when I left the company. Since then they've changed the graphic style of the page, but as far as I've seen, nothing else has changed. They have even left the 1998 Netscape and Internet Explorer logos...

And today I found Google Pages. Basically it does the same, but soooooooo much nicer and interactive and fast and coherent and AJAX and Web 2.0... Just go and play with it! You'll love it.

BirthdayCard Project

Adrian Kosmaczewski

2006-11-05

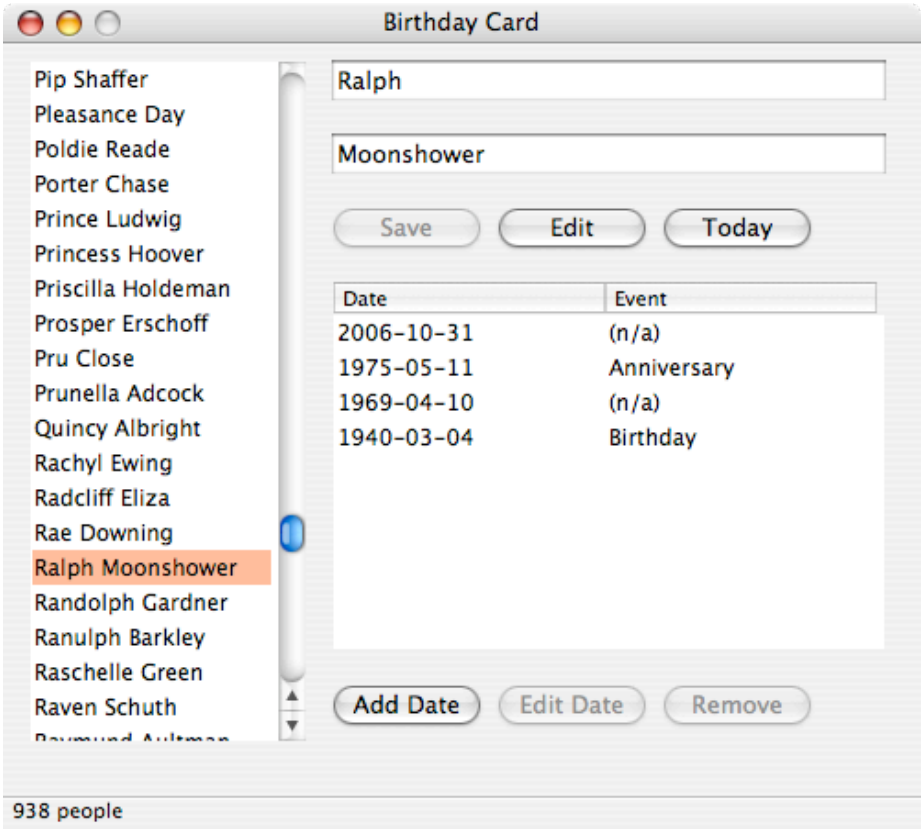
I have just created a project page for the BirthdayCard Project[1]. This was an experimental project for me, to test the combination of several open-source and proprietary technologies such as

- wxWidgets
- SQLite
- CppUnit
- Doxygen
- Eclipse
- Visual Studio
- Xcode
- ISO C++

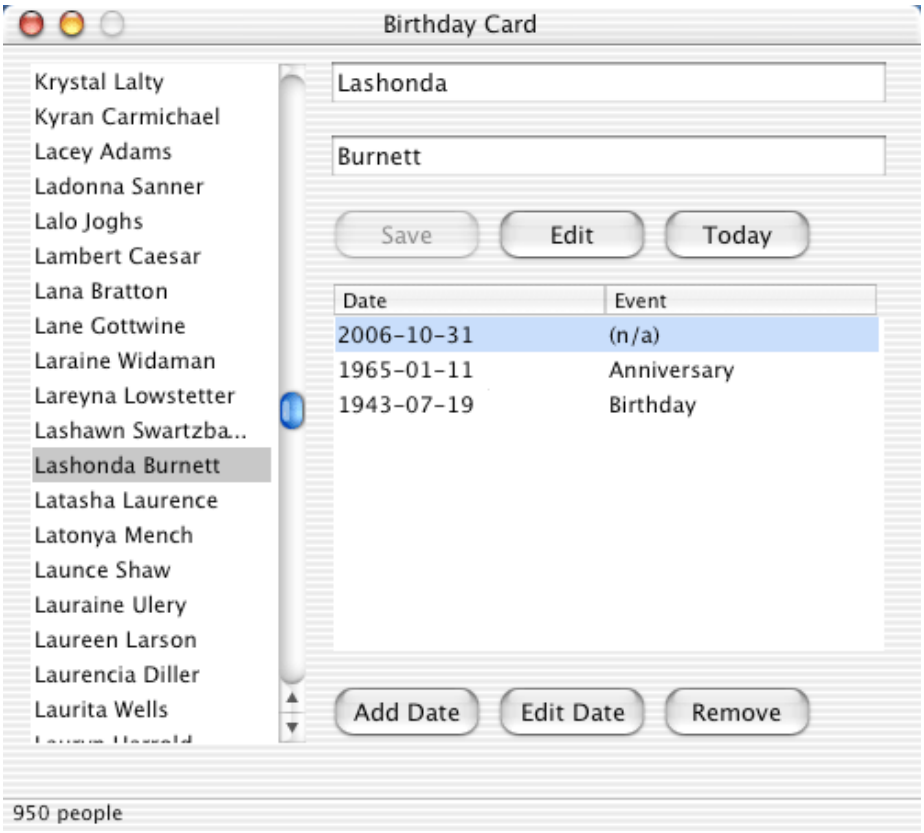
The final result is a small utility that you can use to store birthdays for people you know and care about. You can add as many people as you wish, and for each person, as many dates as needed. It has a layered architecture, in such a way that the underlying "business" classes (namely Person and Date) are reusable in another application (my idea was to add a command line app to show this, but I will do it in a later step). The code is portable, and was tested with 3 different compilers (3 GCC versions, and Microsoft's Visual Studio .NET 2004 one).

Here's a screenshot of BirthdayCard, running on Mac OS X Tiger:

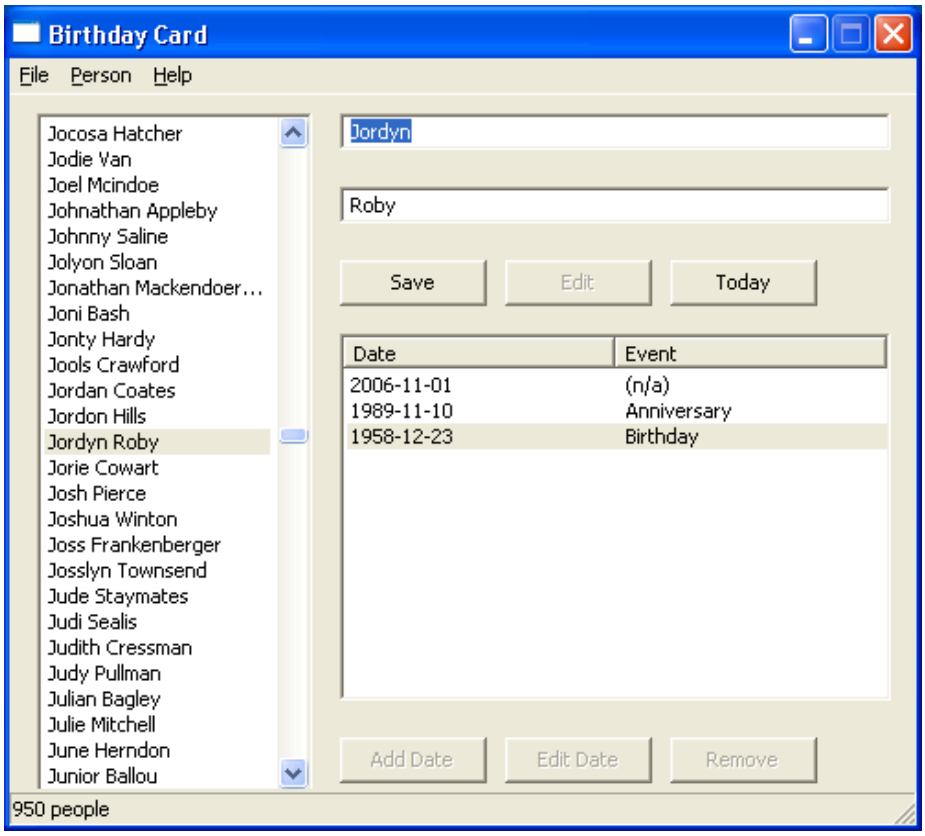
[1] BirthdayCard-1.0-src.tar.gz

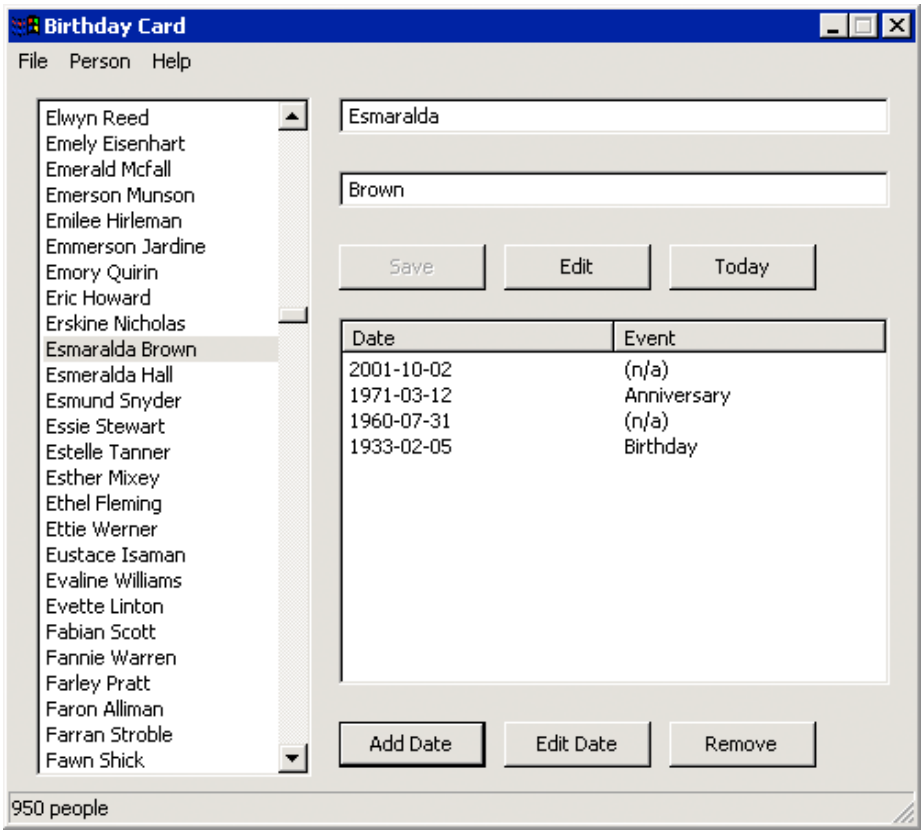


More screenshots in other operating systems:


Birthday Card
Krystal Lalty
Kyran Carmichael
Lacey Adams
Ladonna Sanner
Lalo Joghs
Lambert Caesar
Lana Bratton
Lane Gottwine
Laraine Widaman
Lareyna Lowstetter
Lashawn Swartzba...
Lashonda Burnett
Latasha Laurence
Latonya Mench
Launce Shaw
Lauraine Ulery
Laureen Larson
Laurencia Diller
Laurita Wells
Lashonda
Burnett
Save
Edit
Today
Date
Event
2006-10-31
(n/a)
1965-01-11
Anniversary
1943-07-19
Birthday
Add Date
Edit Date
Remove
950 people


Birthday Card
File
Person
Help
Nikolas Philips
Nita Busk
Nivek Garratt
Noble Fisher
Noelene Dunlap
Nolan Elinor
Norah Yonkie
Noreen Wardle
Nowell Fisher
Nydia Crissman
Oberon Monahan
Odell Stall
Olive Baldwin
Oliver Baxter
Olivia Wynter
Oneida Morland
Opal Kifer
Ora Nabholz
Odell
Stall
Save
Edit
Today
Date
Event
1982-11-03
Anniversary
1939-09-15
Birthday
Add Date
Edit Date
Remove
950 people


Birthday Card
File
Person
Help
Jocosa Hatcher
Jodie Van
Joel Mcindoe
Johnathan Appleby
Johnny Saline
Jolyon Sloan
Jonathan Mackendoer...
Joni Bash
Jonty Hardy
Jools Crawford
Jordan Coates
Jordon Hills
Jordyn Roby
Jorie Cowart
Josh Pierce
Joshua Winton
Joss Frankenberger
Josslyn Townsend
Jude Staymates
Judi Sealis
Judith Cressman
Judy Pullman
Julian Bagley
Julie Mitchell
June Herndon
Junior Ballou
Jordyn
Roby
Save
Edit
Today
Date
Event
2006-11-01
(n/a)
1989-11-10
Anniversary
1958-12-23
Birthday
Add Date
Edit Date
Remove
950 people



I have provided binaries for Mac[2] and Ubuntu[3] (only PowerPC binaries, sorry) and Windows[4]. There is a bundled SQLite database file that contains 950 people, and 2 dates for each person. All in all, it took me 2 weeks to develop the thing, mostly on evenings and weekends.

I have placed the code in the public domain, so that everyone can use it and do what they want with it. The project can be opened with several IDEs, and the code is completely documented for your reading pleasure. Feel free to use this as a template for your own wxWidgets applications, and as always, **I am not responsible of anything that could happen while using this software!** Use it under your own responsibility, learn & teach, and be kind to each other.

In a future article I will post more comments about what I've learnt while creating BirthdayCard.

[2] BirthdayCard-1.0-mac-ppc-bin.dmg.tar.gz
[3] BirthdayCard-1.0-ubuntu-ppc-bin.tar.gz
[4] BirthdayCard-1.0-win-bin.zip

A Couple of Free Books

Adrian Kosmaczewski

2006-11-12

Just for the record:

- Producing Open Source Software[1]
- Become an Xcoder[2]

Happy reading!

Update, 2022-09-02: The “Producing Open Source Software” book has been updated and rewritten in 2021, and it’s available in electronic form in the same URL shown above, just like 16 years ago.

[1] http://producingoss.com/
[2] http://cocoalab.com/cocoalab/developer.php

Virtually Anything

Adrian Kosmaczewski

2006-11-12

What's the hot word these days? **Virtualization**. Even December 2006's issue of Dr. Dobb's Journal[1] talks about it[2]. Even Jason Dixon talks about it[3].

So what's the big deal? Basically the capacity to run different software environments from a single hardware platform, and as such, to streamline operations in diverse and critical fields such as quality, testing, learning, process integration, you name it. Using virtual machines, you can:

- Set up different environments for testing applications simultaneously using different operating systems, either automatically or using ;
- Create standard learning environments for your teams, or your clients, to test new software packages, and to have an easier cleanup/setup cycle before the next training starts;
- Have environments for legacy operating systems, for which you have to maintain compatibility even if you do not own the original hardware any more.

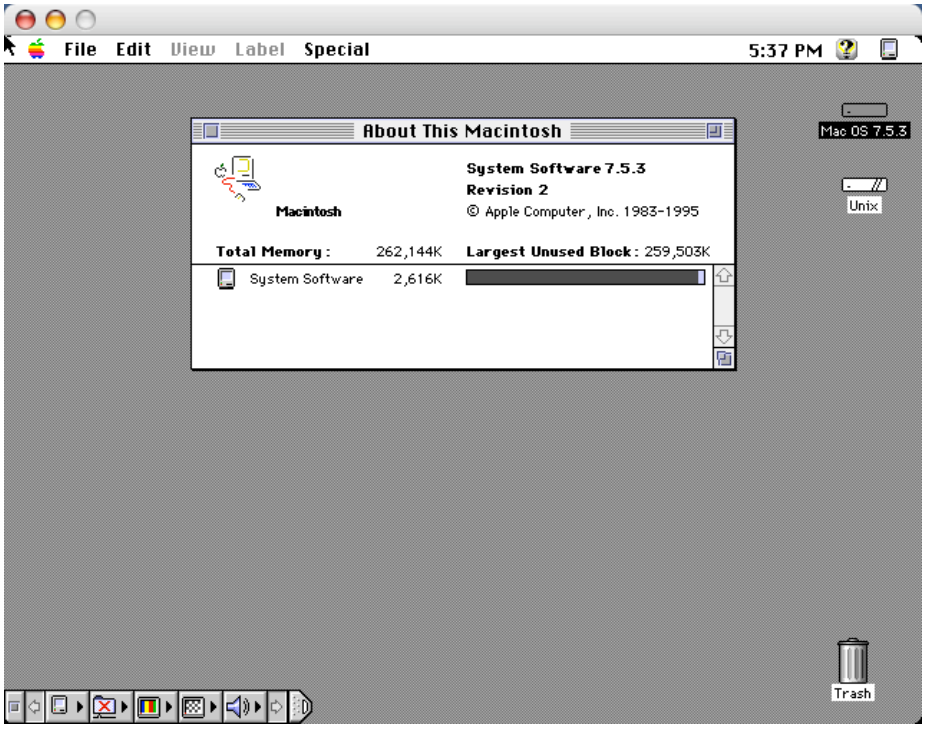
And it all started with emulators; those little applications that eventually grow bigger and bigger and were purchased by big companies with big press releases and all the fanfare. But it all boils down, at some level or another, to this:

[1] http://ddj.com/

[2] http://ddj.com/dept/architect/193501439

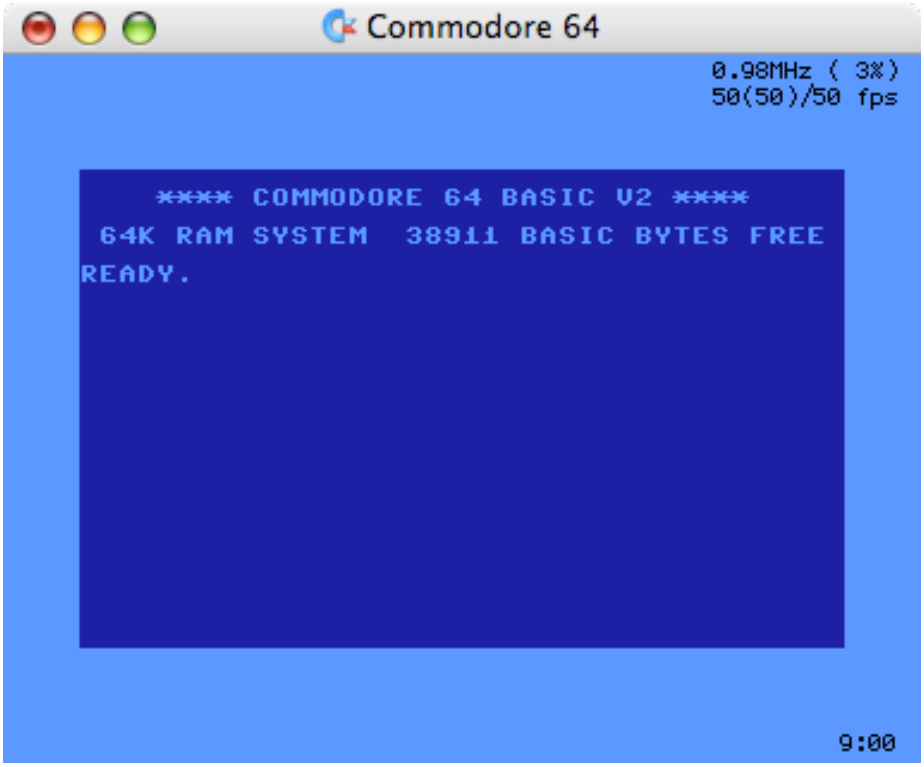
[3] /blog/bsd-is-dying/

Mac OS 7.5.3 (on Basilisk II[4])



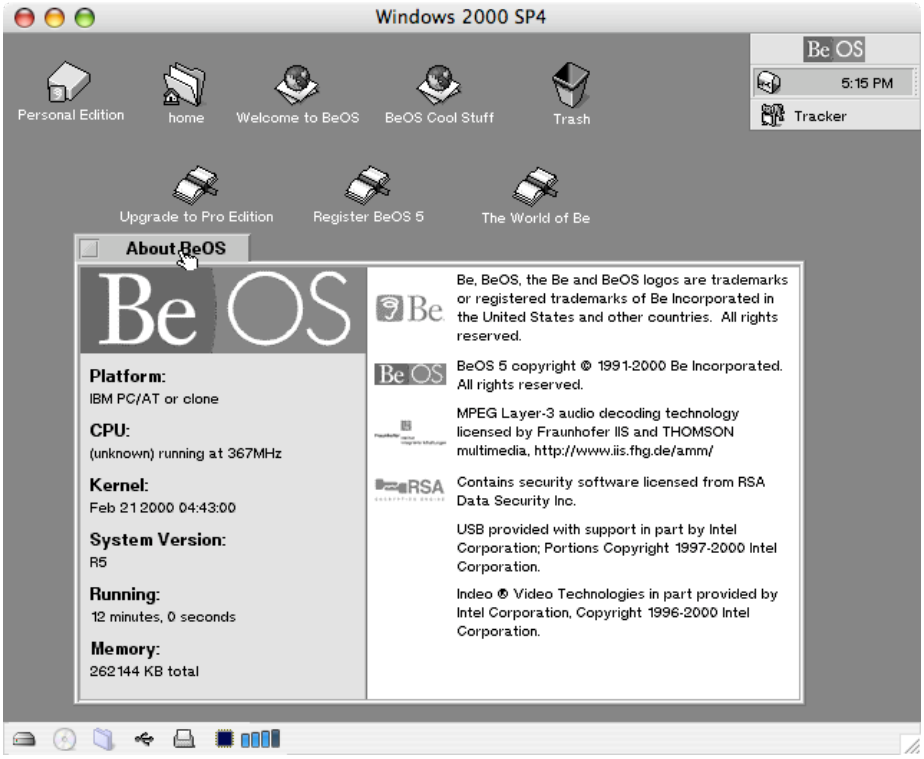
[4] http://gwenole.beauchesne.info/projects/basilisk2/

Power64[5]



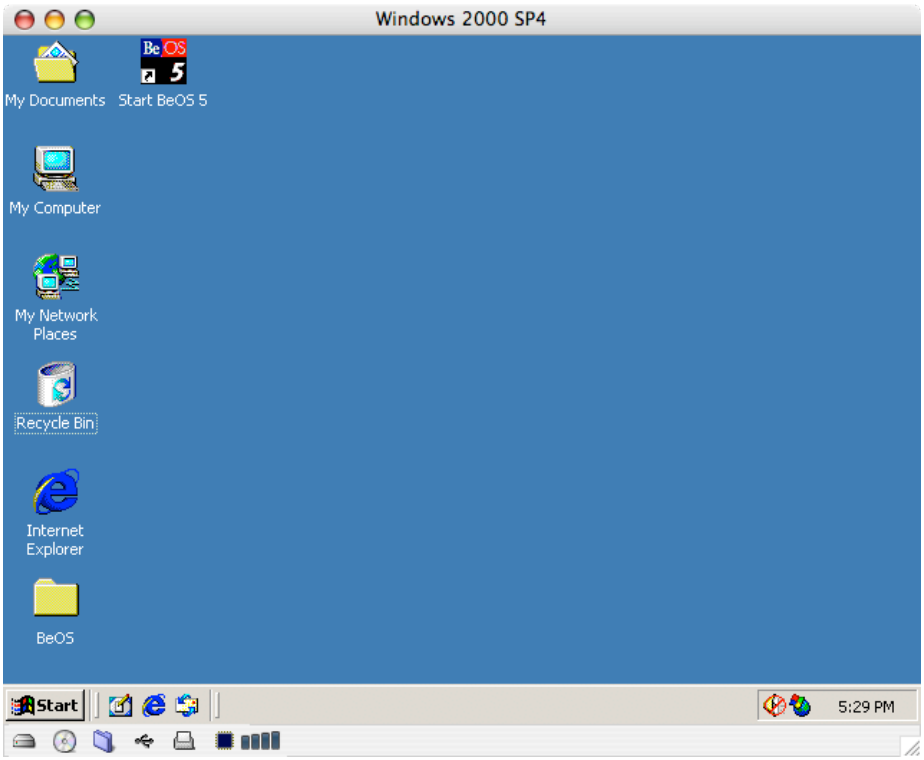
[5] http://www.infinite-loop.at/

BeOS Personal Edition (!?!?, on Virtual PC 7.0.2[6])



[6] http://www.microsoft.com/mac/products/virtualpc/virtualpc.aspx

Windows 2000 SP 4 (on Virtual PC 7.0.2[7])



[7] http://www.microsoft.com/mac/products/virtualpc/virtualpc.aspx

Sobre El Hipertexto

Adrian Kosmaczewski

2006-11-24

tim berners-lee no invento el hipertexto per se; ni siquiera invento su primera aplicacion computadorizada practica.

berners-lee se baso en varios antecedentes historicos, uno de los cuales data de julio de 1945; es un articulo que aparecio en la revista The Atlantic Monthly llamado “As we may think”, de un tal Vannevar Bush (http://es.wikipedia.org/wiki/Vannevar_Bush).

si, hubo miembros de la familia bush que se dedicaban a cosas menos nauseabundas. el articulo en cuestion esta aca: http://www.theatlantic.com/doc/194507/bush

este bush analiza la estructura del conocimiento humano y propone la automatizacion del acceso al mismo, mediante un sistema que nunca se implemento: el **memex** (http://es.wikipedia.org/wiki/Memex)

“Su nombre responde a la contracción de”Memory Expander” de Vannevar Bush. Se trata de un dispositivo, ideado por el autor pero nunca materializado por nadie, en el que se almacenarían todo tipo de documentos. Este dispositivo constaría de una mesa con palancas que buscaría mecánicamente archivos microfilmados que serían posteriormente proyectados en unas pantallas translúcidas. El aparato incluiría también una opción para que el usuario pudiera tomar anotaciones en los márgenes, de manera que el usuario se convierte, a su vez, en autor. Vannevar Bush fue el primero en describir el funcionamiento del Memex en su libro As we may think en 1945.”

en 1960, un tal Ted Nelson (http://en.wikipedia.org/wiki/Ted_Nelson, http://ted.hyperland.com/) lanzo el “proyecto xanadu” (nada que ver con la cancion de olivia newton-john...)

- http://xanadu.com/
- http://en.wikipedia.org/wiki/Project_Xanadu

ted nelson sigue adelante con su proyecto, que consiste en una gran masa de discursos, ideas no realizadas, y una oposicion simple a la world wide web; segun el, la web no es hipertexto: http://ted.hyperland.com/buyin.txt

nadie le da mucha bola.

en 1987, Apple Computer (cuando no) publico la primera version de

un software llamado HyperCard (http://en.wikipedia.org/wiki/HyperCard) que puede decirse, es la primera implementacion directa del concepto de hipertexto en una computadora. el creador de HyperCard es Bill Atkinson, que fue uno de los programadores originales del proyecto Macintosh en 1984; es considerado uno de los padres de la plataforma. el HyperCard se distribuia gratuitamente con los macs, era parte del OS, pero ya no es asi.

el software en cuestion permitia crear "tarjetas" (cards) de informacion, relacionadas entre si, como en una base de datos local, en una computadora dada (no tenia capacidad de conectarse por red a otras bases HyperCard). se podia programar el todo con un lenguaje de programacion "HyperTalk" que es el padre directo del AppleScript.

muchos elementos graficos del HyperCard existen aun hoy en los browsers, ya que tim berners-lee se inspiro de HyperCard para su primer browser; por ejemplo, el cursor de mouse con forma de manito, indicando el hecho de "cambiar de pagina", viene de ahi. Tambien el nombre "home page" esta directamente inspirado, ya que en HyperCard, la primera "card" era la "home card" de la pila ("stack") de cartas.

en 1989 lee invento el protocolo HTTP (que es una capa mas "arriba", mas abstracta que el TCP/IP) y el primer browser "visual" que permitia acceder visualmente a archivos formateados en HTML (otro invento de lee) en una red TCP/IP. estos archivos estan ligados entre si mediante una baliza HTML <a href="...">. el resto es historia. es decir; tim berners-lee invento la primera aplicacion de red basada en el concepto del hipertexto, pero no invento el concepto.

en http://browsers.evolt.org/ se puede bajar el codigo fuente del primer browser: http://browsers.evolt.org/?worldwideweb/ esta escrito en Objective-C, y que es el que se usa hoy dia para hacer aplicaciones en el mac; normal, ya que el mac os x de hoy es una evolucion directa del NeXT de 1989.

de hecho el primer browser de lee se llamo "worldwideweb" pero despues le cambio el nombre a "nexus" para no confundirlo con la red HTTP propiamente dicha.

ahi tambien esta el codigo fuente y un par de imagenes.[1]

[1] http://www.mirrorservice.org/sites/browsers.evolt.org/browsers/worldwideweb/NeXT/screensnap2_24c.gif

Acrobat Reader–Wow!

Adrian Kosmaczewski

2006-12-09

I've just downloaded and installed Adobe Acrobat Reader for the Mac, both in PowerPC and Intel versions, and there's just one thing to say: **Impressive!**

Since Panther I've avoided using the Acrobat Reader on the Mac, since the app seemed clunky and dumb to use; using Preview was simply fast and easy. Adobe Acrobat Reader was just a recompilation of the Windows source code to the Mac; but now, the Reader is a fully-fledged Mac application! It feels excellent, it is blazingly fast, it integrates in Safari, and it looks simply great.

The only drawback? It has the most stupid setup process I've ever seen on a Mac application:

1. Download a DMG file with the installer
2. Run the installer, which downloads the DMG file with the application
3. Install the reader using the installer in the second DMG file

Why not just letting people download the DMG file, open it, and drag the application in the Applications folder, and basta?? In any case, this happens only once, and after that, the application feels really great. This will bring a nice deal of competition in that small field; the Apple PDF reader included in Preview is really nice to use, but I prefer the fullscreen capabilities of the Adobe Acrobat Reader (and yes, I read a lot of PDF files, so this is kind of a power app for me…)

By the way, the book in the screenshot is "Domain Driven Design Quickly", feely available online[1]!

[1] http://www.infoq.com/minibooks/domain-driven-design-quickly

Two New Projects

Adrian Kosmaczewski

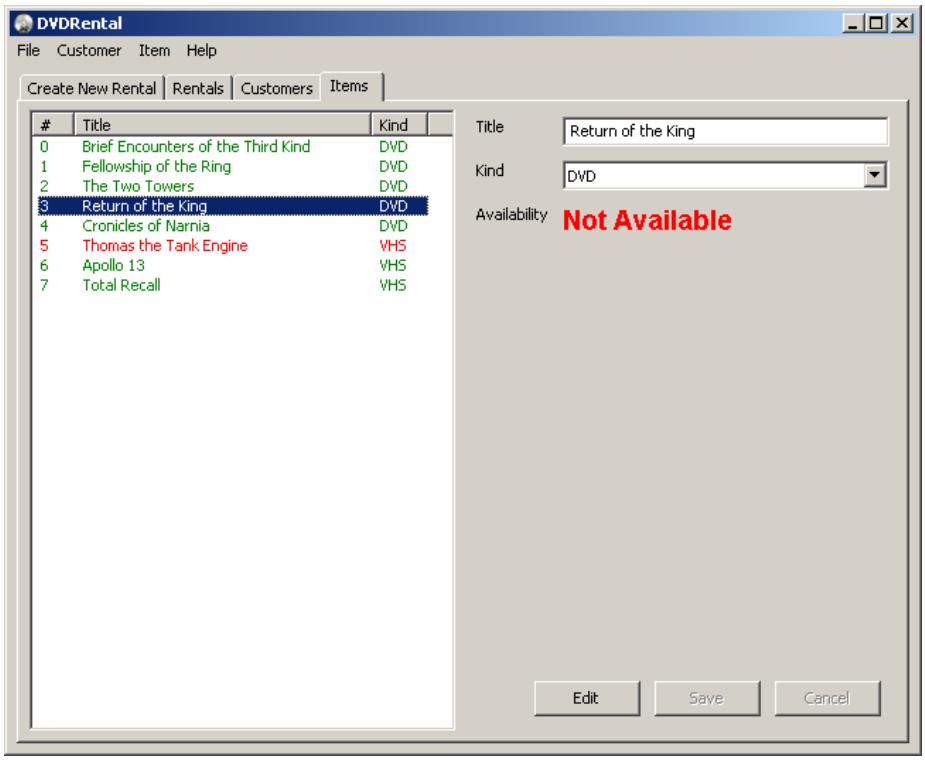
2006-12-14

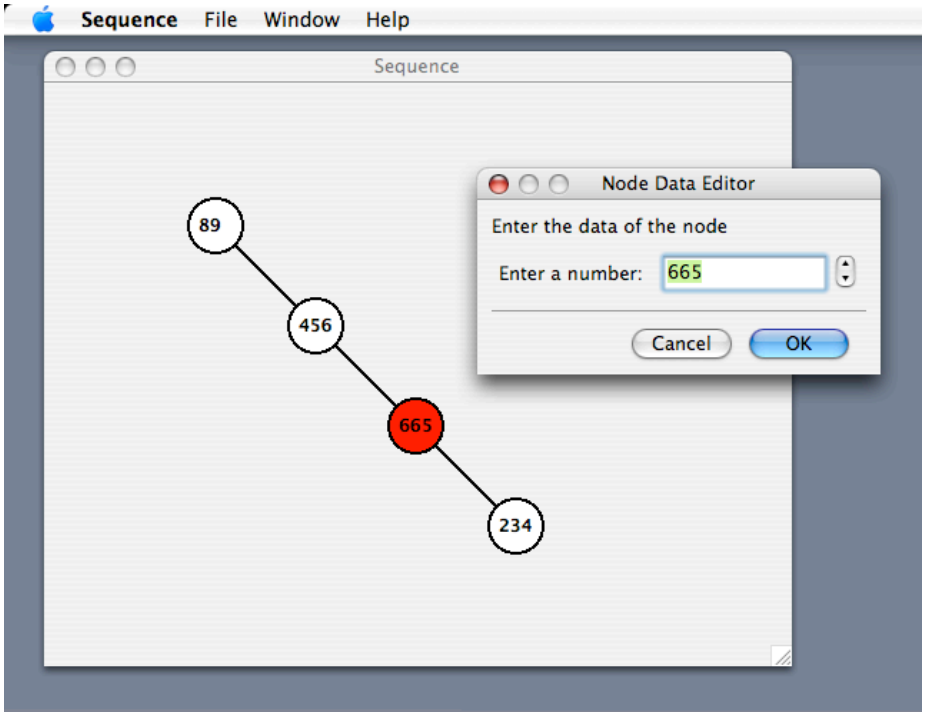
I've just published two more recent wxWidgets projects, always testing new features! These projects should compile without problems with Xcode and Visual Studio.NET 2003 (as always, with wxWidgets, CppUnit and Doxygen).

- **DVDRental**: (source[1], Windows[2], Mac[3]) a small application to manage a videoclub or a shop that rents VHS and DVDs
- **Sequence** (source[4], Windows[5], Mac[6]): another small (even smaller) application to display graphically a linked list on the screen

Feel free to download the code, and as always: this comes without any warranty of any kind... blah blah blah. Do with it what you want and enjoy! :) All comments, as always, welcome.

[1] dvdrental-10-src.zip
[2] dvdrental-10-win-bin.zip
[3] dvdrental-10-mac-ppc-bin.zip
[4] sequence-10-src.zip
[5] sequence-10-win-bin.zip
[6] sequence-10-mac-bin.zip


DVDRental
File Customer Item Help
Create New Rental
Rentals
Customers
Items
Title Kind
0 Brief Encounters of the Third Kind DVD
1 Fellowship of the Ring DVD
2 The Two Towers DVD
3 Return of the King DVD
4 Cronicles of Narnia DVD
5 Thomas the Tank Engine VHS
6 Apollo 13 VHS
7 Total Recall VHS
Title
Return of the King
Kind
DVD
Availability
Not Available
Edit
Save
Cancel


Sequence File Window Help
Sequence
89
456
665
234
Node Data Editor
Enter the data of the node
Enter a number:
665
Cancel
OK

Scrum Software Development Process

Adrian Kosmaczewski

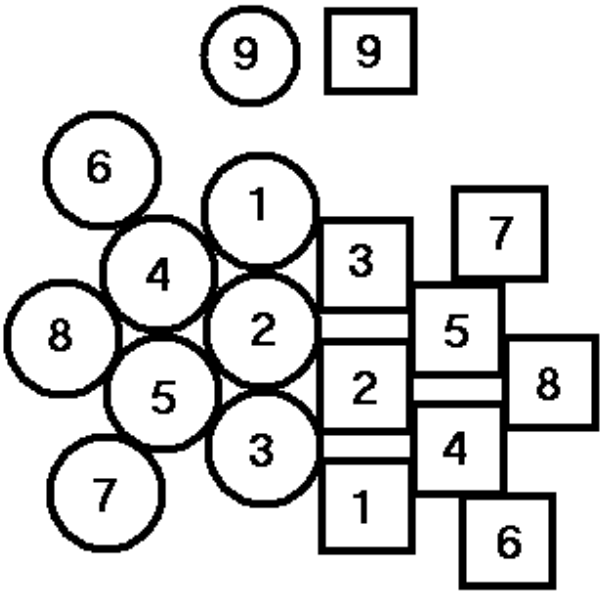
2006-12-28

In this article I give an overview of the **Scrum Software Development Process**. This methodology was first described by Takeuchi and Ikujiro in their 1986 book “The New New Product Development Game”, and was initially meant to manage any kind of product development project. This methodology, for example, has been used in “real” product development projects by companies such as Fuji-Xerox, Canon, Honda, NEC, Epson, Brother, 3M, Xerox and Hewlett-Packard (Schwaber). In the nineties it was adapted for software development projects.

Description

The name “Scrum” is borrowed from the game of rugby, and the following paragraph by Gartner describes the purpose and context of a Scrum in a rugby game:

“A SCRUM is also the name of the formal conglomeration of forwards who bind together in specific positions when a scrumdown is called. It is the basic set formation of rugby and occurs after various minor infringements of the law, when the ball becomes tied up, and other times you’ll learn about later. It is a face-off of sorts and a favorite among forwards. Form and timing are more important than brute strength (although we’ll take some brute strength). A birds-eye diagram might make things more clear:”



“1: Loose Head Prop (sturdy and fearless) 2: Hooker (small, quick, ready to take control) 3: Tight Head Prop (see #1) 4,5: Second Rows (Locks) - (big and strong) 6,7: Wing Forwards (Flankers) - (quick, aggressive) 8: Number Eight (smart, foot and hand skills) 9: Scrumhalf (smart, experienced, quick) – technically not a forward, but the link between forwards and backs - special rules apply to the scrumhalf.”

(Source: Gartner)

The official definition of the Scrum Software Development Process is

> Scrum is an agile, lightweight process that can be used to manage and control software and product development using iterative, incremental practices. Wrapping existing engineering practices, including Extreme Programming and RUP, Scrum generates the benefits of agile development with the advantages of a simple implementation. Scrum significantly increases productivity and reduces time to benefits while facilitating adaptive, empirical systems development.

(Scrum website, 2006)

The most important difference between Scrum and other methodologies is that Scrum does not defines how systems will be developed, using a defined step-by-step guide, but rather how to coordinate the work among team members:

“In this paper we introduce a development process, SCRUM, that treats major portions of systems development as a controlled black box.” (Schwaber)

In this sense, Scrum acknowledges the fact that software development is an inherently chaotic process, complex and often non-deterministic. This lack of determination is given by the following factors (Wikipedia):

- Requirements are never understood at the beginning of the process
- Requirements change
- New technologies bring uncertainty to the processs

As such, Scrum "wraps" existing engineering practices, providing (using Scrum website's own words):

- Complete risk management;
- Ways to manage conflicts;
- Iterative processes in contexts of rapidly evolving requirements;
- Improved communication paths;
- Ways to increase productivity;
- Scalability from small to enterprise-wide projects.

Pragmatics

The Scrum methodology consists of the following phases:

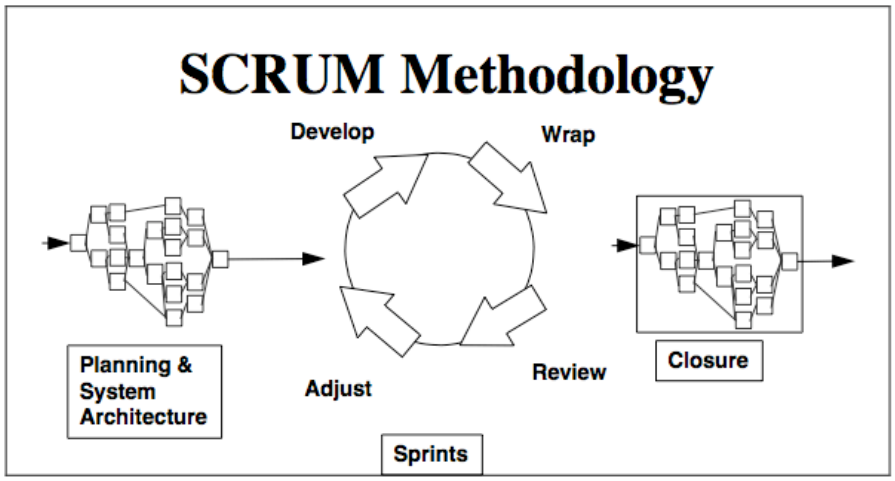


Figure 6 : SCRUM Methodology

1. **Pregame:** this phase holds the planning, and high level analysis and architecture phases. During this phase, the team defines the scope of the project, the milestones dates, deliverables and delivery dates, performs risk assessment and gets approval and funding for the project, as well as defining and reviewing the architecture of the proposed solution. One team member is named the "scrum master", who "leads the Scrum meetings, identifies the initial backlog to be completed in the sprint, and empirically measures progress toward the goal of delivering this incremental set of product functionality." (Rising and Janoff, 2000)

2. **Game:** the development of the system is done in this phase, during what is called a “Sprint”, which is the core and most distinctive element of the Scrum methodology. A sprint is an iterative cycle of one to four weeks, during which developers create the software product. “5 minute daily meetings” are used by the team to quickly coordinate the development tasks for a particular day. Sprints consist of the following high level tasks (Schwaber):

- Develop
- Wrap
- Review
- Adjust

> Daily meetings allowed everyone on the project team to see the status of all aspects of the project in real time. This allowed the collective neural networks of the team’s mind to fine-tune or redirect efforts on a daily basis to maximize throughput. The result was radical alteration of the software development process by allowing sharing of software resources. Development tasks thought to take days could often be accomplished in hours using someone else’s code as a starting point.

(Sutherland, 2004)

After each sprint, the team performs a review meeting, where team and management members participate, needed to evaluate the work done in the previous sprint, and to define and prepare the next one. Three questions define all that is needed to communicate during that meeting: (Rising and Janoff, 2000; Sutherland, 2004)

- What have you completed since the last scrum meeting?
- What obstacles got in your way?
- What do you plan to accomplish until the next scrum meeting?

1. **Postgame:** this phase contains the closure of the project, including the preparation of the deliverables for installation, integration testing and maintenance.

Criticism

As any other methodology, Scrum does not apply to all situations. As Brooks said, “there is no silver bullet”, nor project methodology that applies to all situations. Scrum applies particularly well to projects with a high level of uncertainty (undefined scope, uncanny technologies, new unexplored business models, etc).

Also, in the positive side of the equation, I think that the tight focus on project risks is one of the most important element of the Scrum methodology. Continuously keeping an eye on obstacles, provides a way to react quickly to possible bottlenecks or problems that might

appear during the development process, for which other methodologies do not provide similar quick answers.

On the downside, development teams and project managers should be trained in using the Scrum methodology. I do not think that any project can jump from a formal, "classical" methodology based in defined constraints and feature sets (which usually maps to a more hierarchical team structure), to a model where everyone participates at the same level, providing continuous feedback about possible problems, and communicating in fluid ways. I think that the main problem in adopting Scrum might be a cultural one, rather than a pure methodology one.

As one team leader comments about Scrum, "Give it time to get started before expecting big results. It gets better as the team gains experience" (Rising and Janoff, 2000).

Conclusion

The Scrum Development Process is an interesting example of how important is the acknowledgement of the inherent chaos in the activity of software development, and how important is to communicate clearly and often, particularly when requirements are not defined completely, or when there are other degrees of uncertainty. This recognition leads to a redefinition of the problem, and thus a different viewpoint for managing software projects, in a non-deterministic way. However, as always, a clear definition of the context is needed to ensure the success of the project, choosing the good methodology.

References

Beck, Kent, Thomas, Dave, Fowler, Martin et al., "Agile Manifesto", 2001, [Internet] http://agilemanifesto.org/ (Accessed June 4th, 2006)

Beck, Kent, "What is Extreme Programming?", 1999 [Internet], http://www.extremeprogramming.org/ (Accessed June 4th, 2006)

Frederick P. Brooks, Jr., "The Mythical Man-Month - Essays on Software Engineering, Anniversary Edition", 1995, Addison Wesley, ISBN 0-201-83595-9

Gartner, Lisa, "The Rookie Primer", Radcliffe Rugby Football Club, [Internet] http://hcs.harvard.edu/~radrugby/rookie_primer.html (Accessed June 4th, 2006)

Rising, Linda and Janoff, Norman S., "The Scrum Software Development Process for Small Teams", IEEE Software July/August 2000 [Internet] http://members.cox.net/risingl1/articles/IEEEScrum.pdf (Accessed June 4th, 2006)

Schwaber, Ken, "SCRUM Development Process", [Internet] http://jeffsutherland.com/oopsla/schwapub.pdf (Accessed June 4th, 2006)

Scrum website, [Internet] http://www.controlchaos.com/ (Accessed June 4th, 2006)

Dr. Sutherland, Jeff, “Agile Development: Lessons Learned from the First Scrum”, October 2004, [Internet] http://jeffsutherland.com/Scrum/FirstScrum2004.pdf (Accessed June 4th, 2006)

Takeuchi, Hirotaka, and Ikujiro, Nonaka, “The New New Product Development Game”, Harvard Business Online, [Internet] http://harvardbusinessonline.hbsp.harvard.edu/b01/en/common/item_detail.jhtml?id=86116 (Accessed June 4th, 2006)

Wikipedia, “Scrum (development)”, [Internet] http://en.wikipedia.org/wiki/Scrum_%28development%29 (Accessed June 4th, 2006)

Curso Acelerado De Unix, Linea De Comandos, Y Manejo Basico Del Tema Del Software Open Source, Para Artistas Liberados De Ventanas Colgadas Inutilmente

Adrian Kosmaczewski

2006-12-31

preparate unos mates, imprimi este mail y sentate con tiempo delante de la compu.

asegurate que la mac tenga acceso a internet.

asegurate tambien de haber instalado Xcode; esto es **fundamental** para lo que vamos a hacer ahora; buscalo en el CD de Tiger, en la carpeta “Xcode Tools” (fijate en el attachment xcode.png); tenes un instalador para Xcode, dale nomas sin miedo; instalalo con las opciones “by default”

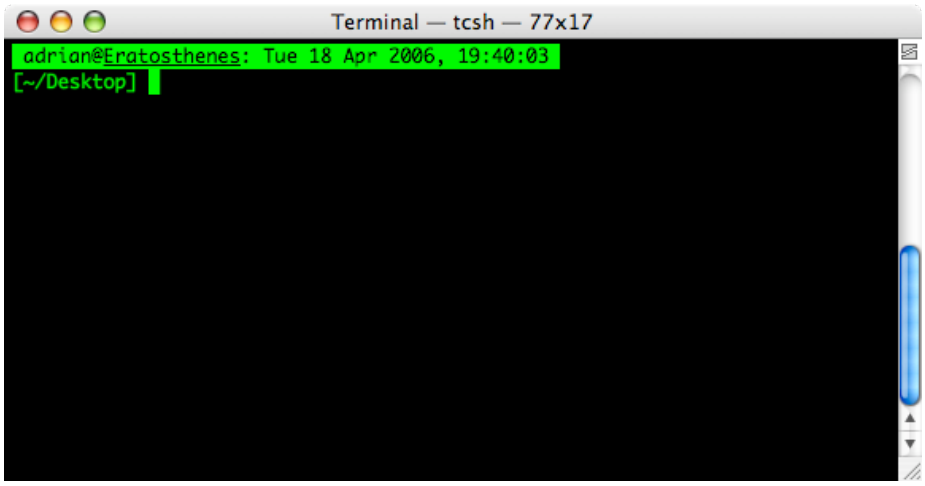
terminal

en el finder, anda al menu “Go”, submenu “Utilities” (tengo la compu en ingles, sori). tambien podes hacer shift+manzana+u para abrir la carpeta “/Applications/Utilities”.

en la ventana que aparece, busca una aplicacion llamada “Terminal”. no, nada que ver con los enfermos. para encontrarla rapidamente, tipea “T”, “E” y “R” en el teclado rapidamente y veras que el sistema te la selecciona.

abri “terminal”

veras una ventana que aparece con un cursor y una tilde. en mi caso, la ventana se parece a esto:



si, mi compu se llama “Eratosthenes”. otro dia te explico.

en tu caso tendra otros colores y otro texto, pero eso no importa ahora. un dia te explico como se hace para personalizar el asunto; en las preferencias de Terminal podes cambiar los colores (yo uso este formato desde hace años, pero cada cual a lo suyooooooo).

carpetas

el “~” o tilde significa “home”, es decir, en el mac es un “shortcut” hacia “/Users/adrian” en mi caso, o “/Users/pirulo” en el tuyo, si tu nombre de usuario es pirulo, cosa que desconozco.

la “/” barra vertical significa la raiz del disco duro de donde buteo el sistema, en el mac es el icono del disco arriba de todo a la derecha en el finder.

para ubicar otros discos, usas “/Volumes”, por ejemplo, si conectas un disco externo a la mac, y el disco se llama “pirulo” en el finder, en la linea de comando lo podes ir a buscar con un “cd /Volumes/pirulo”, asi de facil.

cd es “change directory”. fijate que si, estando en “~” haces “cd Desktop” estaras en el Desktop del usuario conectado actualmente. es el mismo desktop que ves encima del fondo de pantalla, ni mas, ni menos. ahora veras como es esto.

podes hacer “**cd /Applications**” y veras todas las aplicaciones si tipeas “**ls**” (ele ese); el comando ls te “listea” los contenidos de la carpeta.

`cd ~` te lleva a tu home de vuelta

hasta ahi vamos bien?

hace `cd ~/Desktop`

y ahora hace esto: `mkdir test`

no le des cualquier nombre, usa “test” ya que usare esta carpeta durante todo el tutorial; veras que en tu escritorio aparece una carpeta nueva llamada, precisamente, “test”. asi de facil.

ahora metete en esa carpeta nueva, con un **“cd test”** tendria que alcanzar. para salir de ahi, “**cd ..**” (cd-espacio-dos puntos) y con eso volvemos al escritorio. basico. y con “ls” podes ver los contenidos. basico.

tambien se pueden borrar carpetas, enteras, incluso con contenido pero eso es para otra vez; no borres la carpeta test, la usaremos dentro de un rato.

para limpiar la pantallita, tipea “**clear**”. tambien podes usar “manzana+k” pero no es estandar unix, solo para mac.

editar texto

ahora hace “**vim archivo.txt**” (si, “vi” como el pasado del verbo ver en brasilero, con eme al final)

aparece un editor de texto, medio raro pero que es lo mas.

ahora tenes que seguir escrupulosamente las indicaciones que te voy a dar, ya que no hay menuse.

tipea “**i**” (la letra i) que significa “insertar” (linda actividad, si las hay)

ahora empeza a escribir texto, lo que se te ocurra

cuando termines de escribir, apreta la tecla “**esc**” (escape, arriba de todo a la izquierda en el teclado)

ahora para guardar el texto y salir, tipea “**:x**” (dos puntos x)

mira en el finder, tenes un “archivo.txt” en tu carpeta “test” en el escritorio. el finder mantiene la coherencia visual.

vim es un editor de texto para el que hay que conocer los comandos, no queda otra; ya te mandare mas comandos. por ejemplo, si entras en vim sin haber dado el nombre del archivo antes y queres guardar lo

tipeado, tipea “esc” para pasar en modo comando y “**:w archivo.txt**” para “escribir” (write) el contenido en el archivo.txt. si queres salir sin escribir nada, tipea “**:q!**” (dos puntos - cu - signo de exclamacion) que es un “file/exit/no quiero guardar gracias”, pero todo en 3 caracteres.

parco el creador de vim, si los hay.

y asi volves a la sempiterna linea de comandos.

por ahora hace lo siguiente (tipea la secuencia de comandos siguiente); basicamente vamos a tu “home” y vamos a crear un archivo especial:

```
cd ~
vim .vimrc
```

tipea “i” para empezar a entrar texto, y tipea lo siguiente:

```
set number
set autoindent
syntax on
set tabstop=4
set showmode
set showcmd
set mouse=a
set nowrap
```

tipea “esc” y “:x” para guardar la cosa en el disco.

el archivo “.vimrc” es un archivo oculto; en unix todo archivo que empieza por un punto es oculto. y es el archivo de preferencias de vim (los archivos de preferencias en unix terminan en “rc”, que no se que quiere decir).

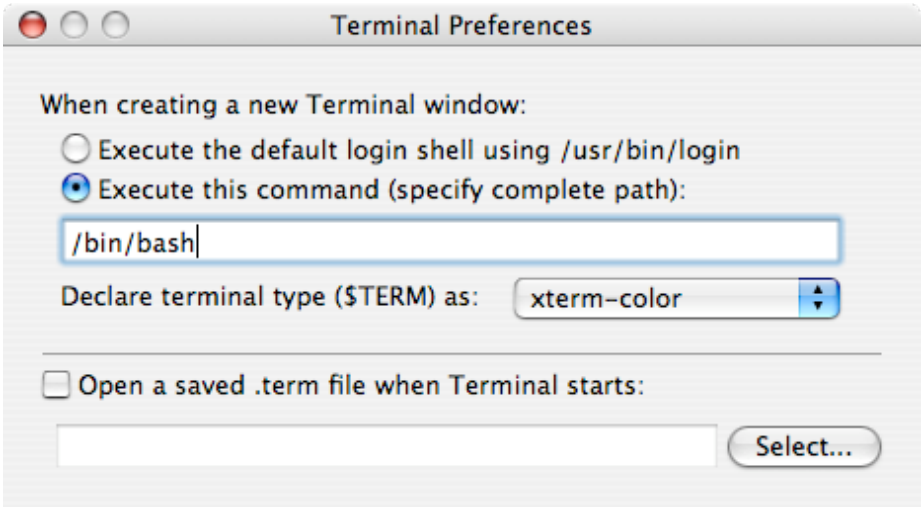
ahora abri de vuelta el .vimrc con el comando “vim .vimrc” y veras la diferencia.

preferencias de terminal

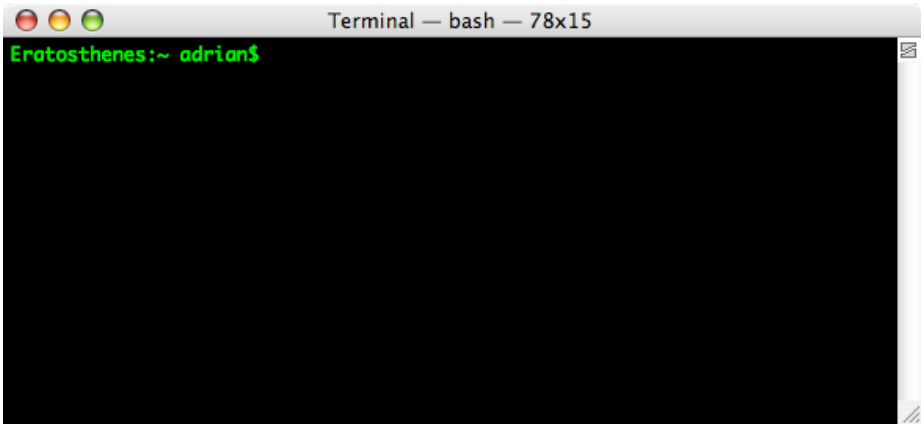
en unix hay varios programas que hacen linea de comando: los tres mas conocidos son “sh”, “csh”, “bash” y “tcsh” (yo uso este ultimo). el programa “terminal” es simplemente una aplicacion que “contiene” uno de estas lineas de comando. nada mas. el “sh” significa siempre “shell” o “cascara”, ya que se dice que la linea de comando “envuelve” el OS.

los comandos entre las diferentes lineas de comando cambian ligeramente, pero los comandos estandar son identicos. cuanto comando, comandante.

para nuestro ejercicio, vos fijate de que estas usando “bash”:



cuando abris una ventana de terminal con bash, se parece a esto



el mundo open source

ahora tipea lo siguiente:

```
vim ~/.bash_login
```

y adentro, tipeas “i” para entrar en modo de insercion, y tipea lo siguiente (o copy pastealo)

```
export PATH="/usr/local/bin:/usr/local/sbin:$PATH"
```

y apreta “esc” y luego “:x” para grabar el archivo y salir de vim.

si, ya se, es criptico, no te preocupes que ya iras entendiendo. por lo pronto, como veras, vim, en su uso basico, no es tan complicado, es siempre lo mismo, y esta bueno conocer un editor de texto (que ademas, es una masa).

lo que acabamos de hacer es un programa; para ejecutarlo, tipea .

~/.bash_login (es decir, y ATENCION con esto: punto-espacio-tilde-barra-punto-bash_login, ok?)

no dice nada la pantalla, todo bien.

ahora movete a tu carpeta “test”, la que creaste un par de etapas antes: “cd ~/Desktop/test”

tipea los siguientes comandos, uno despues de otro, sin chistar; es decir, tipeas un comando, y esperas a que termine de ejecutarse. ahi ejecutas el otro, y asi sucesivamente (lo mejor es que hagas copy paste)

```
curl -O ftp://ftp.gnu.org/gnu/readline/readline-5.1.tar.gz
tar xzvf readline-5.1.tar.gz
cd readline-5.1
./configure --prefix=/usr/local
make
sudo make install
cd ..
```

bueno, habras notado de que despues del comando “sudo make install” el sistema te pide el password; tipealo y listo.

“sudo” es “super user do”, es decir, es pedirle al sistema unix que haga algo extremadamente sensible, que solamente el usuario administrador total, superusuario, hiperman, supradios de la computadora, puede hacer: el usuario root, que aunque no lo veamos, siempre esta.

felicitaciones, acabas de compilar tu primer proyecto open source.

lo que has hecho ahi arriba es lo siguiente:

1. bajar de una url (curl) un fichero zipeado (formato .tar.gz) del proyecto gnu, llamado readline, version 5.1.
2. luego lo descomprimiste
3. te metiste en la carpeta resultante de la descompresion
4. configuraste la compilacion segun tu propia computadora
5. hiciste (“make”) el binario mediante compilacion
6. y lo instalaste como superusuario (“make install”)
7. finalmente, volves a la carpeta superior en la jerarquia carpetistica.

“make” es un programa que sirve para automatizar compilaciones complejas, con muchas etapas y dependencias; la idea es que “configure” genera un “makefile” que es ejecutado por make. asi funciona esto. y el binario final, esta perfectamente adaptado a tu computadora. esta compilado para ella. mejor, imposible.

vamos por otro; el lenguaje ruby, ultima version disponible, la 1.8.4:

```
curl -O ftp://ftp.ruby-lang.org/pub/ruby/1.8/ruby-1.8.4.tar.gz
tar xzvf ruby-1.8.4.tar.gz
cd ruby-1.8.4
```

```
./configure --prefix=/usr/local --enable-pthread --with-
readline-dir=/usr/local
make
sudo make install
cd ..
```

como siempre, asegurate de copiar rigurosamente los comandos, preferentemente copy/paste.

sigamos, para instalar “rubygems”:

```
curl -O http://rubyforge.org/frs/download.php/5207/rubygems-
0.8.11.tgz
tar xzvf rubygems-0.8.11.tgz
cd rubygems-0.8.11
sudo /usr/local/bin/ruby setup.rb
cd ..
```

y ahora, vamos a cambiar de comando, para instalar “rails”, el framework de creacion de aplicaciones web, que se instala mediante rubygems (comando “gem”):

```
sudo gem install rails --include-dependencies
```

bueno, ya con esto tendras suficiente. otro dia vemos para mysql ;)

mira en la carpeta “test” en tu escritorio, tenes el codigo fuente de las tres aplicaciones open source, para que veas como son, y los makefile y todo eso, y ahora sabes usar un editor de texto, asi que te invito a que tipees esto y que veas que bueno esta “vim”:

```
vim ~/Desktop/test/ruby-1.8.4/array.c
```

es un archivo de codigo fuente en lenguaje C, del proyecto ruby, escrito por el GENIO ABSOLUTO DE LA PROGRAMACION MUNDIAL, BARRILETE COSMICO DE QUE PLANETA VINISTE matsumoto. y con colores, lineas, todo lo que hace falta para leer codigo comodamente.

espero no haberte mareado :)

Curso Acelerado De Subversion (Primera Parte)

Adrian Kosmaczewski

2007-01-01

tenes que aprender subversion. por lo pronto tenes que instalarlo en tu mac, lo cual es facilisimo ya que hay un instalador que lo hace por vos, y que tenes que bajarte de aca:

http://www.codingmonkeys.de/mbo/Subversion-1.4.2.pkg.zip (por las dudas las ultimas versiones estan siempre disponibles en http://www.codingmonkeys.de/mbo/)

es un zip que contiene un pkg de instalacion; tenes que tener tiger, o panther 10.3.9 como minimo para que ande. cuando lo instales no veras ninguna diferencia en tu compu, ya que se instala en /usr/local/bin/svn. (/usr es un folder escondido en el disco, que es parte de todos los unix).

subversion es un sistema cliente-servidor de control de versiones; no es el primero, pero si es uno de los ultimos y mejores. por "control de versiones" quiero decir que subversion mantiene en una base de datos todas las versiones de cada archivo de cada proyecto y te permite ver que cambios hubo entre la version 2 y la 3. no hay software que se haga seriamente hoy dia sin algo asi. por "cliente-servidor" quiere decir que el software tiene una arquitectura simple, donde un cliente muy ligero se conecta a un servidor, que es donde estan los datos guardados. por "servidor" no implico una computadora separada, ya que en realidad el servidor es un daemon andando en una compu (daemon = programa sin interfaz grafica, llamado "servicio" en windows, o mas generalmente proceso). pero tambien puede ser otra computadora, y subversion habla muchos protocolos de red, algunos securisimos como ssh y otros mas especificos como svn, pasando por http y acceso directo al sistema de archivos, si fuese menester, para leer los datos del **repositorio**.

porque ese es el nombre que se le da a una base de datos de versiones: **repository**, o **repositorio**. en muchos tutoriales veras que dicen "ponga su base de datos en `C:\repos` y 'repos' viene a ser eso, repositorios.

hay muchos productos que hacen esto del control de versiones: IBM tiene Rational ClearCase (lo use en alguna empresa en la que trabaje

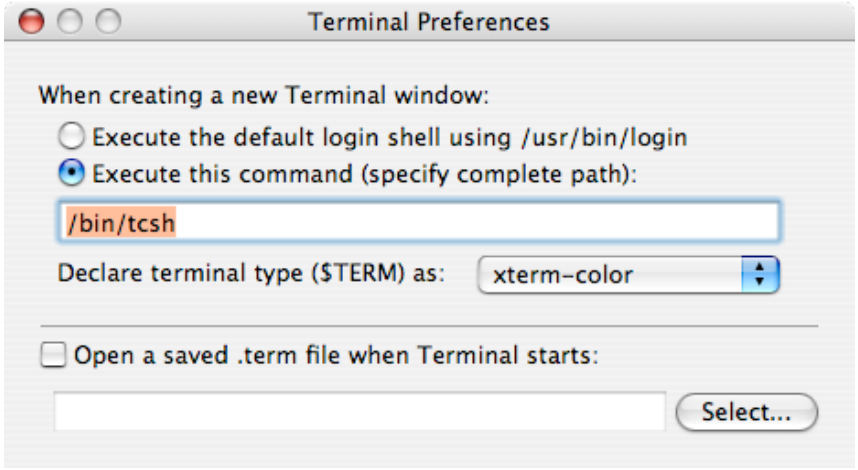
antes), Microchot tiene Visual SourceSafe (de lejos el peor de todos, tambien lo sufri) y ahora Team System (que no he usado ni usare, sale 20 lucas la licencia, estan de la gorra). Tambien esta SourceGear Vault, CVS (que es open source y reconocidisimo mundialmente) y tantisimos mas.

pero subversion es el mas nuevito (version 1.0 del 2004) y esta inspirado en CVS (Concurrent Version System). para usar subversion comodamente hay que estar comodo con la linea de comando, ya que subversion (svn para los amigos) es, en el mejor espiritu unix, un utilitario de linea de comando (primariamente). ahora hay muchos programas que permiten manejar svn visualmente, y gratis, asi que tranqui que una vez que tengas la parte teorica del asunto, usar la consola grafica te sera mas facil. subversion anda en cuanta plataforma se te cruce; obviamente tambien ubuntu, mac (desde la version jaguar 10.2 que tengo en la ibook), windows (desde la version 98 SE, creo), tanto para servidor como para cliente.

por ahora hace esto:

- instala el paquete Subversion-1.3.1.pkg (te pedira el password ya que instala en /usr)
- abri una terminal.app
- tipea “svn” en la linea de comando; si no ves un texto que dice “Type ‘svn help’ for usage.”, quiere decir que no logra ubicar svn. si es asi:

1. tipea “where is svn” (asi nomas) y te dira donde esta; ojo! yo uso “/bin/tcsh” como aplicacion de terminal, y no “/bin/bash”, en las preferencias de terminal.app:



2. luego anda a tu home (cd ~) y edita el archivo de preferencias de tcsh (vim .tcshrc) y por ahi agrega estas lineas:

```
alias svn "/usr/local/bin/svn" (con las comillas)
alias svnadmin "/usr/local/bin/svnadmin"
alias svnserve "/usr/local/bin/svnserve"
```

3. cerra el archivo (esc y luego :x como te explique)
4. cerra la ventana de terminal y abri una nueva. ahora si, si haces “svn” te tiene que decir el texto Type ‘svn help’ for usage.”.
5. lo de bash y tcsh tiene que ver ya que el comando “alias” es propio de tcsh, no se cual usas vos.

una vez que hagas todo esto, avisame que seguimos con el tutorial mas tarde. te mostrare como crear un repositorio en tu compu, en local, como yo uso varios ya, meter codigo fuente ahi adentro, y despues volver a sacarlo, pero esta vez versionado y todo.

he aqui la version en español del libro de o’reilly sobre subversion que es la referencia absoluta: http://svnbook.red-bean.com/index.es.html

en la proxima te cuento como usar subversion.

Curso Acelerado De Subversion (Segunda Parte)

Adrian Kosmaczewski

2007-01-02

antes que nada fijate que todo lo que te explique anteriormente anda.

empecemos.

primero armate una carpeta con archivos varios, en tu escritorio. pone imagenes, archivos php, javascript, html, carpetas con mas cosas adentro, lo que se te ocurra, con cuantos subniveles que quieras. cualquier disposicion estara bien. lo unico que te pido es que llames esa carpeta "proyecto", simplemente (aunque podria ser cualquier nombre, yo usare ese nombre en este texto, y asi sera mas facil seguirme; pero podria llamarse "pirulo" tranquilamente).

tambien te pedire que pongas un archivo "readme.txt" en esa carpeta, para mostrarte como se manejan los cambios en los archivos. asegurate que ese archivo tenga ya un poco de texto. tambien agregate un archivo mas, que se llame, "forradas.txt", con algo de texto adentro.

bueno, ahora tenes tu proyecto listo, tenes la primera version de tu laburo (estamos imitando un ritmo de trabajo tipico con subversion). podrias estar escribiendo un nuevo libro, un software, dibujando con el GIMP, lo que quieras. cualquier "proyecto" puede ser versionado con subversion. es decir, escribiste un par de parrafos, tenes una idea de la estructura del libro, o ya tiraste un par de lineas de codigo y pensas que esto puede evolucionar. entonces, tenes ganas de "versionarlo" para poder volver atras en caso de hacer un moco, o para explorar diferentes maneras de hacer algo, de manera "paralela". por ejemplo, yo uso subversion para el material de mi master; de esta manera, si laburo en la portatil, o en la mac de mi escritorio, siempre tengo la ultima version de lo que hago a disposicion, sin importar donde hice la ultima modificacion.

entonces ahora vamos a crear el repositorio. yo los tengo en ~/Repositories, pero en realidad pueden estar en cualquier lugar. los pongo ahi para poder bacapearlos facilmente. supondre que los pondras en ~/Repositories tambien. **asi que create una carpeta "Repositories" en ~** (si no te acordas que corno significa la tilde, fijate aca). yo usare esa carpeta durante este texto, y tambien asumire que tu

nombre de usuario es **xxxxxx** (obviamente, cuando veas **xxxxxx** en el texto aqui debajo, reemplazalo por tu usuario en el mac).

crear repositorios es tan facil y barato en terminos de espacio disco que se recomienda activamente tener un repositorio por proyecto, en vez de varios proyectos en el mismo repositorio. es lo mas logico y despues veras por que es altamente recomendable.

abri terminal.app y mandate con un "cd ~/Repositories" (si tipeas "cd ~/Repos" y apretas la tecla TAB el resto de la palabra se completa automaticamente).

ahora estas en la carpeta donde estaran tus repositorios. para crear un repositorio, momento clave, tipeas algo facilisimo:

```
svnadmin create Proyecto
```

ahora si te fijas, ya sea con la linea de comando o con el Finder, veras que tenes unos 4 MB de carpetas creados adentro de una carpeta llamada "Proyecto". esa carpeta es tu repositorio. ahi adentro estan las bases de datos que guardaran las 1002423 versiones de tu proyecto. el nombre "Proyecto" es algo libre, podes hacer "svnadmin create Piruleta" y te hara otro repositorio con el nombre Piruleta.

fijate que en el repositorio "Proyecto" hay un archivo llamado Readme.txt; adentro dice claramente que:

```
This is a Subversion repository; use the 'svnadmin' tool to examine
it. Do not add, delete, or modify files here unless you know how to
avoid corrupting the repository.

Visit http://subversion.tigris.org/ for more information.
```

bueno, como veras, dice que no toques nada ya que podrias romper todo. la carpeta del repositorio, en si, nunca se toca directamente SINO a traves de los comandos svn y svnadmin. de otra manera, no se hace nada. bueno, en realidad, si, haremos algo, muy cortito y simple, para que puedas usar facilmente el repositorio.

adentro del repositorio hay una carpeta "conf" y adentro veras un archivo "svnserve.conf"; abrilo con cualquier editor de texto (vim, textmate, cualquiera) y veras que todas las lineas tienen numerales "#" delante; son comentarios. para que ande el repositorio (por medida de seguridad, un repositorio svn no anda per se, hay que "habilitarlo", que es lo que hacemos ahora) tenes que sacar el numeral de las lineas 8 y 12, y modificar la linea 12 para que todo quede asi:

(extracto)

```
### Visit http://subversion.tigris.org/ for more information.

[general] ### These options control access to the repository for
unauthenticated ### and authenticated users. Valid values are
"write", "read", ### and "none". The sample settings below are the
defaults. anon-access = write # auth-access = write ### The
```

```
password-db option controls the location of the password ###
database file.  Unless you specify a path starting with a /, ### the
file's location is relative to the conf directory.  ### Uncomment
the line below to use the default password file.  # password-
db =
passwd
```

la linea 12 explicitamente da derechos de escritura a los usuarios anonimos. asi de facil. despues veremos la parte de seguridad. pero por ahora, veras que nos alcanza asi para que veas el funcionamiento. como veras, al tener todo comentado, el repositorio es seguro, ya que nadie puede hacer nada con el. esto es tambien parte de la filosofia unix, asi como tuviste que habilitar php en apache usando el httpd.conf. como veras, siempre la misma idea; a priori, las puertas a los intrusos estan cerradas, pero se pueden abrir facilmente con un poco de laburo.

me seguis hasta ahora? bueno, seguimos. ahora tenemos que crear, adentro del repositorio, una estructura de base que es recomendada por los creadores de subversion, pero no obligatoria, y que es de crear, adentro del repositorio, tres carpetas donde vamos a meter la informacion: “trunk”, “tags” y “branches”. por ahora te digo que el proyecto en si ira dentro de “trunk” (creo que significa “cajon” o “baul” en ingles). en “tags”, por ejemplo, guardaremos las diferentes versiones del proyecto para poder encontrarlas mas rapidamente (tipo “version 1.0”, etc). “tags” significa “etiquetas”. “branches”, finalmente sirve para crear lo que se dice “forks” (“tenedores”), es decir, proyectos alternativos, generalmente usados para estudiar prototipos o diferentes soluciones a un problema, de manera paralela, sin obstruir el trabajo ni el contenido de “trunk”. es algo potentisimo.

PERO ATENCION! trunk, tags y branches seran carpetas “virtuales”, y no de las que se crean con el Finder; las vamos a crear desde la linea de comando de la manera siguiente:

```
svn mkdir file:///Users/xxxxxx/Repositories/Proyecto/trunk -
m "Creacion de trunk"
svn mkdir file:///Users/xxxxxx/Repositories/Proyecto/branches -
m "Creacion de branches"
svn mkdir file:///Users/xxxxxx/Repositories/Proyecto/tags -
m "Creacion de tags"
```

como veras, svn usa los nombres de comandos tipicos unix, en este caso mkdir, para hacer operaciones simlares dentro del repositorio. el parametro “-m” toma un string de texto como parametro, que acompaña la modificacion hecha al repositorio, para que uno pueda saber que paso. svn exige que cada vez que se hace una modificacion, se use el parametro -m para indicar de manera textual lo que se hizo. este texto tiene que ser coherente, ya que se usa para saber quien hizo que, cuando, donde, etc.

si no pones el texto, svn se queja:

```
svn: Could not use external editor to fetch log message; consider
setting the $SVN_EDITOR environment variable or using the -
-message
(-m) or --file (-F) options svn: None of the environment variables
SVN_EDITOR, VISUAL or EDITOR is set, and no 'editor-
cmd' run-time
configuration option was found
```

tambien fijate de la sintaxis `file:///` con 3 (tres) barras oblicuas, ya que se trata de una URL, y no de un mero camino de acceso. el URL indica a svn el protocolo que tiene que usarse para acceder al repositorio (http, file, svn o svn+ssh son los mas comunes, nosotros usaremos todos salvo http).

ahora tenemos:

- tu proyecto
- el repositorio listo para recibir los datos.

llego el momento de la verdad; vamos a meter la carpeta “proyecto” del escritorio en tu repositorio:

```
cd ~/Desktop
svn import proyecto file:///Users/xxxxxx/Repositories/Proyecto/trunk -
m "Import inicial"
```

y listo! todo el contenido de la carpeta ~/Desktop/proyecto esta ahora en subversion. totalmente versionadito y guardadito. para eso sirve el comando “svn import”.

y ahora podes tirar a la basura la carpeta /Users/xxxxxx/Desktop/proyecto.
esto es algo que puede parecer raro, pero tu carpeta /Users/xxxxxx/Desktop/proyecto no esta versionada, sino que fue usada para alimentar el repositorio. lo cual no es lo mismo. para seguir laburando en tu proyecto de manera versionada, tenes que tener una version versionada (sic), generalmente llamada “working copy” del proyecto:

```
cd ~/Desktop
svn checkout file:///Users/xxxxxx/Repositories/Proyecto/trunk proyecto
```

con este comando, le estamos diciendo a svn que nos de una copia versionada del proyecto, sacandolo de la carpeta “trunk” del repositorio file:///Users/xxxxxx/Repositories/Proyecto, en una carpeta local que se llame “proyecto”. y listo! ahora tenes una version lista para trabajar en tu escritorio.

bueno, ahora si te digo, que lo jodido, ya paso. las operaciones que quedan son las mas simples. lo que hicimos hasta ahora es algo que se hace generalmente una sola vez cada tanto, y ahora viene el trabajo diario.

modificate un par de lineas de readme.txt; cualquier cosa, lo que se te ocurra. poco importa, con cualquier editor. tambien create un nuevo archivo adentro de la carpeta proyecto, que se llame piruleta.txt, y pone un poco de texto adentro.

con esto estamos simulando cambios diarios dentro del proyecto. es el trabajo cotidiano. se modifican cosas, se agregan otras. por ahora no borres nada, ya te mostrare como se hace.

ahora vamos a la linea de comandos a ver que ha pasado en el proyecto despues de un dia de trabajo:

```
cd ~/Desktop/proyecto
svn status
```

y vemos esto:

```
?      piruleta.txt
M      readme.txt
```

como veras, ? significa que hay un archivo nuevo, que svn desconoce, y otro que fue modificado. los demas archivos no aparecen en esta lista. cuando haces svn status en una copia local versionada, ves los cambios hechos hasta ese momento. muy interesante. ahora vamos a “agregar” piruleta.txt al repositorio:

```
svn add piruleta.txt
```

y de paso, vamos a borrar “forradas.txt”, que despues de todo son solo forradas:

```
svn delete forradas.txt
```

y si pedimos el status,

```
svn status
```

vemos que

```
A      piruleta.txt
D      forradas.txt
M      readme.txt
```

donde A = add (agregar); D = deleted; M = modified; mas facil, imposible.

bueno, ya laburaste mucho por hoy, asi que vamos a subir los cambios al repositorio. si “checkout” es para obtener una copia de trabajo versionada, “commit” es para mandar los cambios al repositorio:

```
cd ~/Desktop/proyecto
svn commit -m "Hecho algunos cambios..."
```

en la jerga se dice “commite unos archivos hace un rato” para indicar que fueron puestos en el repositorio. como veras, siempre aparece el comando -m para indicar un texto que cuente lo que paso.

si ahora haces

```
svn status
```

veras que svn no dice nada; quiere decir que tu version es la ultima. para ver las cosas que se hicieron, podes hacer un

```
svn log readme.txt
```

y eso te muestra lo siguiente:

```
------------------------------------------------------------
--------------
r5 | xxxxxx | 2006-05-20 16:37:03 +0200 (Sat, 20 May 2006) | 1 line

Hecho algunos cambios...
------------------------------------------------------------
--------------
r4 | xxxxxx | 2006-05-20 16:25:10 +0200 (Sat, 20 May 2006) | 1 line

Import inicial
------------------------------------------------------------
--------------
```

obviamente las fechas cambian, depende de cuando vos hiciste las cosas. ahi ves la necesidad de

- tener un repositorio por proyecto (para que no se mezclen los mensajes de log) y
- poner un texto descriptivo cuando se realiza una modificacion del repositorio

pero bueno, saber que hubo un cambio en readme.txt entre la version actual (5) y la anterior (4) no te ayuda mucho; que habra cambiado?

```
svn diff -r 4 readme.txt
```

y eso muestra:

```
--- readme.txt  (revision 4)
+++ readme.txt  (working copy)
@@ -1,13 +1,8 @@
The files in this folder contain the definition of the MySQL database
used to store the information of the application. Its structure
-is as simple as possible, allowing minimum flexibility.
-NOTES:
-1) InnoDB tables are used to support FOREIGN KEY relationships between tables
-
-
+esta parte la agregue durante la ultima revision
2) "timestamp" type columns do not hold values of type NULL, even the table
definition says so; following MySQL documentation, "timestamp" columns
that receive a NULL store the current date and time automatically.
```

(obviamente tu texto sera distinto) pero lo importante son los signos “+” y “-” que indican las lineas de texto que se agregaron y se sacaron entre la version actual (la “working copy”) y la revision 4 (la “r” es de revision).

como veras, esto es simplemente tocar la punta del iceberg. svn es un mundo aparte, pero permite cosas realmente esotericas.

bueno, creo que por hoy es bastante; obviamente hemos visto solamente la mecanica para un solo usuario; svn es usado en varios

proyectos conocidos en el mundo open source (mira la lista aca, es realmente impresionante), y ahora tambien sourceforge lo ofrece para los proyectos que alli se alojan (antes era solamente cvs).

imaginate que alguien, por ejemplo yo, modifica el repositorio durante la noche; como haces para ponerte al dia vos? bueno, a la mañana siguiente, o cuando te aviso por e-mail, haces simplemente un

```
svn update
```

con este simple comando, svn va a buscar la ultima version del repositorio, y te pone tu copia local al dia. si hubieses hecho modificaciones sin commitarlas, pero que fueron modificadas en el repositorio (una situacion mas que comun), svn se encarga de "mezclar" convenientemente las modificaciones remotas con las tuyas, y si no puede hacerlo, te pregunta. como veras, es tremendo. ese proceso de mezcla se denomina **merge** en la jerga de los sistemas de gestion de versiones.

finalmente, existe un proyecto open source, hecho en php, que se llama websvn que permite ver los contenidos de un repositorio con un browser. yo lo tengo instalado en casa para ver rapidamente mis repositorios, muy practico.

te dejo con un

```
svn help
svn help mkdir
svn help commit
svn help checkout
svn help update
svn help add
svn help import
svn help diff
```

para que veas las distintas opciones que tiene svn... para cada uno de los comandos que aprendiste hoy.

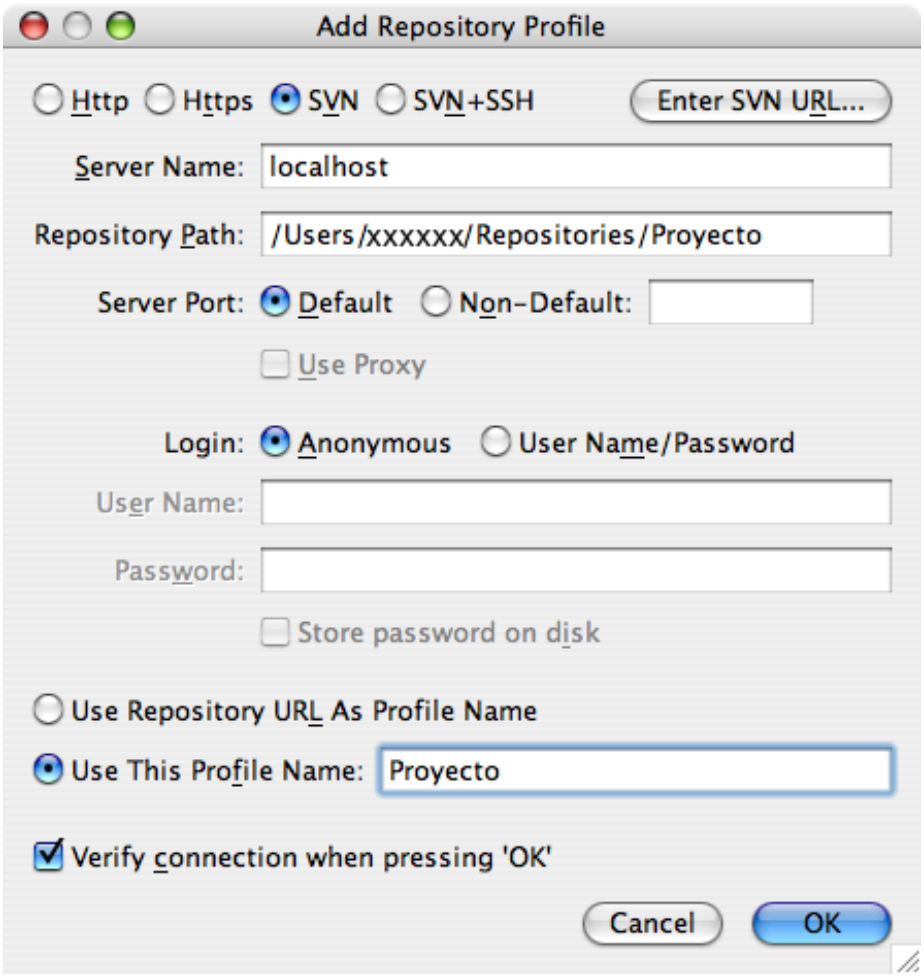
que te parece? subversion hoy dia esta integrado a visual studio, eclipse y a xcode, de manera muy natural, para poder "mirar" repositorios de manera simple y asi obtener copias de trabajo, enviar modificaciones, etc. muchos proyectos estan siendo "migrados" de cvs, sourcesafe y otros sistemas a subversion, ya que es tecnicamente muy superior, gratis, open source y activamente mantenido. subversion fue escrito en C, usando el "apache portable runtime" como framework para acelerar el desarrollo.

bueno, bajate SmartSVN y tendras un cliente subversion escrito en java que esta muy bueno; configuralo y paseate por tu repositorio (necesitas java 1.4.1 para que ande...)

si no podes usar smartsvn, es normal, tenes que largar primero esto en la linea de comandos

```
svnserve -d
```

y despues configura el repositorio asi:



aqui arriba estas pidiendo conectarte mediante el protocolo “svn” al repositorio, por eso necesitas el daemon svnserve (que es el que escucha el puerto 3690, el puerto de subversion) para asi poder usar el protocolo svn.

creo que con esto tendras para divertirte un rato..... :)

una vez que le tomes la mano a esto, con linea de comando o no, veras que lo necesitas absolutamente. para cualquier tipo de proyecto, y no solo para software.

About Microsoft “Standards”

Adrian Kosmaczewski

2007-01-10

An excellent article about how to fool everyone to believe that your specification is a… standard:

> This is a running criticism I have of Microsoft’s Office Open XML (OOXML). It has been narrowly crafted to accommodate a single vendor’s applications. Its extreme length (over 6,000 pages) stems from it having detailed every wart of MS Office in an inextensible, inflexible manner. This is not a specification; this is a DNA sequence.

I want one

Adrian Kosmaczewski

2007-01-10

Anyone knows when are these coming to Switzerland?

[1]

[1] http://www.apple.com/iphone/

Some .NET Code

Adrian Kosmaczewski

2007-01-23

I just updated the Projects subsection of this site with some .NET code that I wrote, between 2003 and 2006:

- OrugaSystem[1]: The OrugaSystem project came to my mind first as an HTML <–> RTF converter, but later evolved into a generic transformer framework; it was created with Visual Studio .NET 2003, and it runs under the .NET Framework, version 1.1.
- .NET 2.0 Samples[2]: I created these applications for illustrating some aspects about .NET 2.0: ADO.NET 2.0 Providers, ClickOnce, BackgroundWorker, C# Generics, FileSystemWatcher component, StringBuilder, Unit Testing, and the MultiView component. There is a little bit of everything, from command-line applications to Windows Services and fully distributed architectures. These applications were written using Visual Studio 2005, and run under the .NET Framework, version 2.0.

Have fun! :)

[1] orugasystem.zip
[2] dotnet-2-samples.zip

About Java

Adrian Kosmaczewski

2007-01-26

Are you scared of Java language change? Why?

> Java 5 introduced generics amongst many other items. Unfortunately, generics are probably the most troublesome change that has been made to Java. With over 400 pages in the official generics FAQ attempting to explain weird corner cases, we know something went wrong.

Update, 2023-05-12: Internet Archive snapshot[1] of the original article.

[1] https://web.archive.org/web/20121028130532/http://blog.joda.org/2007/01/are-you-scared-of-java-language-change_7632.html

How to Grab or Capture Your Screen With Cocoa

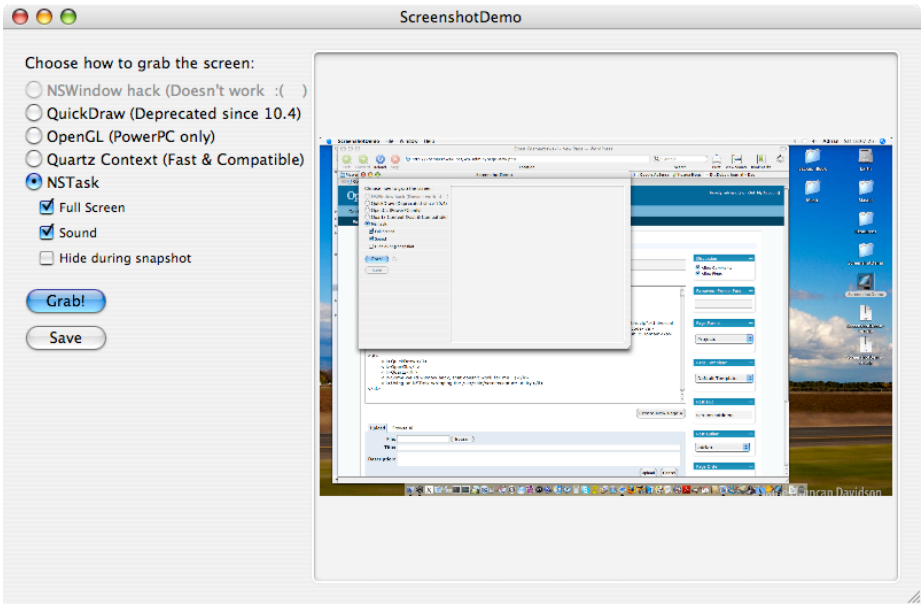
Adrian Kosmaczewski

2007-01-27

Lately I got curious to know how could I grab the entire desktop of my computer, and save it into a file, or display it into an NSImageView component. I started to look around on the web and discovered that:

- There's no direct support for that in Cocoa
- There's a lot of different ways to do it, both supported and unsupported, cross-processor and not, easy and complicated

I have found several useful resources in my quest, like this one[1], this other one[2], and finally this one[3]. But what I wanted most was a complete application to play with, so what I did is to put all the different implementations I've found in one single application, called "ScreenshotDemo":



[1] http://www.cocoabuilder.com/archive/message/cocoa/2006/4/23/161734

[2] https://web.archive.org/web/20071008195756/http://www.cocoadev.com/index.pl?ScreenShotCode

[3] https://web.archive.org/web/20080509110907/http://www.sticksoftware.com/developer/Screensnap.m.txt

Amazingly, the approach that seems the ugliest turned to be the most appropriate, that is, using an NSTask instance wrapping the **/usr/sbin/screencapture** utility. With it, the application feels lighter, easier to maintain, in the true, purest Unix style: using a collection of small utilities, all chained one to the other, is better than having an overbloated tool that does everything, but just bad.

You can just download the (universal) binaries and the source code from the ScreenshotDemo Project page[4] (sources[5]).

By the way, for those that would like to do the same in C#, like me :) just check out this code[6]. It isn't much easier in .NET, as you can see ;)

And last but not least, here's how to do it in wxWidgets[7], explained by Julian Smart[8], the creator of this incredible library.

[4] screenshotdemo-bin.zip

[5] screenshotdemo-src.zip

[6] https://web.archive.org/web/20070204105914/http://www.c-sharpcorner.com/UploadFile/perrylee/ScreenCapture11142005234547PM/ScreenCapture.aspx

[7] https://web.archive.org/web/20070527224159/http://lists.wxwidgets.org/archive/wx-users/msg62690.html

[8] https://web.archive.org/web/20070608054543/http://www.anthemion.co.uk/julian.htm

Now Wait a Second:

Adrian Kosmaczewski

2007-01-27

There was a time where programming in Java was funnier than it is today[1].

Back in 1997 I published in my own web page my first Java applet. It was a calculator. That was way cool; I was able to share with the visitors of my site a whole program: not only the source code, but the whole thing, working, available for anyone to use, no matter the browser, no matter the platform, no matter whe geographical location.

You can play with that applet here[2][3]. It still works on my MacBook and on my G5, almost 10 years later, running under the Java 1.5 virtual machine. I am simply amazed (It might not be the most interesting code you'll ever see, nor the most beautifully architectured one, but it did what I wanted it to do... so I consider it a success. Thankfully I see things different now).

So there was a time were Java was cool, new, easy to learn, the way to go. And here's a story to show how cool it was.

In 1998 I was in Buenos Aires, working in a dot-com company, programming and designing a website; we were using Windows NT 4 servers, and as such the web server was IIS, and the whole thing consisted of "classic" ASP pages, programmed using VBScript. Talk about technology; Java was non-existent in this environment. However, it was for me the basis for a small solution that proved really useful, during many years.

Our CEO came one day with an idea; in our website forum, the users should be able to upload images showing the products being offered. The images were uploaded on the server, and then they were showed beside the text of the offer. Rather simple requirements, right? Well it turns out that those images could have any size; I do not mean in bytes, but in pixels. Said like this, it does not seem like a big deal, but actually this poses a basic design problem. Since the width and height of the images are arbitrary, placing them correctly on the web

[1] http://www.jroller.com/page/scolebourne?entry=are_you_scared_of_java

[2] ../creative-processes/CalcApplet/CalcApplet.html

[3] **Update, 2023-05-12:** Chrome and Microsoft Edge users can run the calculator applet mentioned in this article using the CheerpJ Applet Runner[4] extension.

page, with the right proportions, brought some problems to the technical team (that was just 2 people: my colleague and I). And so we realized that being able to know the width and height of the uploaded images was the missing link to solve the problem (and to keep our CEO happy).

I started to look for external components fulfilling the task, but without success. Apparently there were no available components with the required feature: being able to return the width and height of images uploaded to the server. As such, I decided that this was an interesting thing to tackle, and I decided to create such a component by myself.

That's the "advocating inventor" thing that you can read in my Personal DNA profile[5]. I love to do this kind of investigations.

First, since PNG files were not (yet) handled correctly by the mainstream browsers of the time (Internet Explorer 4 and Netscape Communicator 4), we decided that supporting GIF and JPEG files was more than enough for our needs. The challenge, then, was to parse the files looking for that information.

The first thing to do, is detect the file format. And for that, you just can't trust the file extension. A malicious user could upload a ZIP, an EXE or a DOC file, with a misleading extension, and your component would fail. The idea then is to open the file, regardless of the extension, and find a way to guess the file format reading the bytes inside. This is usually easy, since the first three bytes of a GIF file are numbers 71, 73 and 70: the letters, G, I and F, as you might guess. JPEG files start with the numbers 255 and 216. Easy to read.

Once you have the format, just choose the right algorithm for getting the width and height information. For GIF files, this is not a big deal: the width is stored in bytes 6 and 7, and the height in bytes 8 and 9. This explains why the maximum width and height of a GIF file is 65,535 pixels, since 65,535 = 216 - 1. Which makes for a really big image anyway.

But the tricky part was yet to come; JPEG files have a really, really weird way to store the width and height. It is just not stored in a fixed place, like in the case of GIF files; the position of this information is stored in the file, somewhere, and then, using that information, you must "jump" to that part of the file. Anyway, this is how the algorithm goes:

- Scan the file, from the beginning, three bytes at a time, until you find a sequence like this: 255, 195 and 192 (yes, it's weird)
- Once you find that sequence, stop searching; from the byte that contains 192 (with address "x"), the height is stored in bytes x + 5 and x + 6, and the width in bytes x + 7 and x + 8.

As you can imagine, when I found the algorithm, after a couple of

[5] http://www.personaldna.com/report.php?k=WrnyDNhmwTSHHPX-GM-ADDCA-3c48

days searching for it, I was extremely happy! I must thank the guys from the comp.graphics.algorithms newsgroup, who pointed me to the right resources[6] at that time.

Looking back in time, I asked myself, why did I do it in Java? Well, first of all, because it was a fine and strict programming language to work with, that I had in my work computer and at home, and in which I could create a stable implementation, in a language easy to read. The environment dealt with memory management for me, was stable, free, and the end result ran quite fast in every computer I tested it.

Of course, our web server being a Windows box, and not having a Visual J++ license at work, we decided to rewrite it in Visual Basic 5, and create a COM component with it. This way, we could install it in our server, and it ran gracefully for more than 2 years before being replaced.

Later on, since the code did not handle the new JPEG 2000[7] format, we came accross some files that could not be handled properly by the component. Nevertheless, it proved useful and handled, almost without flaw, during 2 years, 24/7, lots of GIF and JPEG files that our customers uploaded to the server. Maybe it failed to recognize a couple of files, over tens of thousands, in a live environment.

Actually, now that I think of, I realize that at the time we completely overlooked the fact of the ominous GIF licensing thing of Unisys[8], but since

> only the software firms who sell the enabling software for profit would be expected to secure a licensing agreement from Unisys.

I think that we were outside of the problem. Besides, their patent rights are no longer what they used to be:

> The Unisys patent expired on 20 June 2003 in the USA, in Europe it expired on 18 June 2004, in Japan the patent expired on 20 June 2004 and in Canada it expired on 7 July 2004. The U.S. IBM patent expired 11 August 2006, The Software Freedom Law Center says that after 1 October 2006, there will be no significant patent claims interfering with employment of the GIF format.

(Source: http://www.gnu.org/philosophy/gif.html)

And now, you can download the code from my imgsize project page[9]. Feel free to play with it! As usual, do it at your own risk. All comments welcome, of course! And I hope that nobody will get sued for this, either :)

---

[6] http://groups.google.com/group/comp.graphics.algorithms/browse_thread/thread/66a7096fb00e802a/4776e8f41c03cb1f

[7] http://www.jpeg.org/jpeg2000/

[8] http://lpf.ai.mit.edu/Patents/Gif/unisys.html

[9] imgsize.zip

Migration

Adrian Kosmaczewski

2007-01-28

I came accross this interesting posting[1] on The .NET Addict's Blog[2]. It is interesting to read since it comes at a time where "traditional" Microsoft developers[3] get interested[4] in other technologies, such as Ruby on Rails, Cocoa or Linux. And the same can be said about Java, where lots of luminaries like James Duncan Davidson[5] (the creator of Ant and Tomcat) started moving towards other technologies (in his case, this happened at the beginning of this decade even).

Traditionally Microsoft has had a very strong relationship with their developers[6], since they are the ones that create the applications, that ultimately drive the sales of Windows, whatever the version. But more and more, developers are discovering new ways to do things. And when developers open their minds, it means that the next generation of project managers, architects and CIOs will change the shape of industry; this is a slow process, but a steady one. At the end, some technologies will prevail, others will fail.

There is a big change going on. But what are the common traits of the technologies that attract developers?

I will concentrate my comments on programming languages, which are, after all, the basic tool we use every day in our job. Of course, IDEs, code generators and frameworks usually come after, bringing productivity and ease of use, but they usually depend on the underlying programming language.

The first common characteristic of the languages that are attracting developers is dynamicity. If you compare Ruby, JavaScript, Objective-C, Smalltalk and Lisp, with C++, Java, and the .NET languages, you see that the new languages allow you to dynamically interact with objects on the runtime environment; it's not reflection-by-API (like in the case of .NET or Java) or reflection-by-macros (like you need to do in C++) but direct reflection, embedded in the language and the

[1] http://dotnetaddict.dotnetdevelopersjournal.com/leopard_techtalk.htm

[2] http://dotnetaddict.dotnetdevelopersjournal.com/

[3] http://www.softiesonrails.com/2006/8/21/rails-is-officially-life-changing

[4] http://peterwright.blogspot.com/2006/09/good-bye-microsoft-pete-has-now-left.html

[5] http://blog.duncandavidson.com/

[6] https://www.youtube.com/watch?v=SaVTHG-Ev4k

runtime. Just ask the object about its class, add some methods to it (even if you do not have the source code of the object!), create the API that best suit your code, and make your code as readable as a Paulo Coelho book.

Another benefit of dynamic languages is the fast write-compile-run cycle; in most of today's fashionable languages, the lack of a compiler helps delivering faster, not only at release time, but also at maintenance time (which far longer).

Even in the case of Objective-C, which is a compiled language after all (thanks to the GCC suite), most of the references are resolved at runtime, so you have faster compile-link cycles actually; Objective-C is an extremely interesting language, that has a feature that I haven't seen anywhere else: the **id** type[7]. id is not NSObject (the root class of Cocoa), but rather a "placeholder" type (and more similar to a void* type, actually, but easier to use), which allows a variable to point to whichever object, no matter the type; this allows you to be extremely dynamic, like in scripting languages, while at the same time being able to optimize your code using static references.

However, nowadays, the only factor where static typing bring great value, is in the case of IDEs: autocompletion and "intellisense" are all empowered by strong typing, and thus, when you hit the "dot" character, you get the list of methods of the object you are working on. Usually performance was also considered a byproduct of static typing; but what about nowadays? Yes, compiled languages perform faster, but economical and market reasons are making dynamic languages more interesting everyday; code is much more than performance. **Time-to-market, maintainability and readability are getting more important today.** And, besides, faster processor speeds make runtime languages an interesting and fast option, after all.

The second common denominator is automated memory management. It just frees your mind and time to think about the problem in hand, instead of having to worry about dangling pointers and memory leaks. Actually, this was the missing element in Objective-C, but Apple will be incorporating it in version 2.0.[8] This makes Objective-C an even more interesting language to developers: you get performance (you can use static typing if you want, which also empowers the Xcode IDE), you get flexibility (you can use pure dynamic typing if you want), and you get automated memory management (which lowers the overall workload of a developer). Will Objective-C's garbage collector bring more enterprise developers to the Mac OS X platform? I think yes.

The third common factor is the reduced lines of code that you write in this language, compared to the statically-typed ones (this is kind of a corollary of the second factor I've mentioned above). And simply

[7] http://www.onlamp.com/pub/a/mac/2001/05/04/cocoa.html
[8] http://www.apple.com/macosx/leopard/xcode.html

stated, less code means less maintenance costs, faster bug resolution, and faster time-to-market. These are important factors today; and one of the marketing factors that make Ruby on Rails so popular. This will not mean the end of statically-typed and system languages; particularly C++ and Java have a strong future in the finance industry, where they still are very strong.

In conclusion, I think that the trend is very well set, and that the migration is taking place. Developers are migrating towards dynamic, garbage-collected, and more discrete languages. The benefits are there, and now, the performance too. Remember when EasyJet[9] started? Nobody imagined that a small airline selling on the web could overtake the industry. It is, as Paul Graham[10] says, the Power of the Marginal[11]. And now we'll see that in the programming world as well.

[9] http://www.easyjet.com/
[10] http://paulgraham.com/
[11] http://www.changethis.com/26.03.PowerMarginal

This Year's Programming Languages

Adrian Kosmaczewski

2007-01-30

Trying to keep my promise of learning a new programming language every year[1], I have identified a couple of candidates for 2007:

- Lisp[2]
- D[3]
- Haskell[4]
- Erlang[5]

In any case, I want to get my hands dirtier with functional programming. And also, in the meanwhile, getting used to the new versions of old friends, both bringing new and interesting features:

- C# 3.0[6]
- Objective-C 2.0[7]

If you have any suggestions, about other programming languages, do not hesitate to tell me in the comments below!

[1] /blog/a-new-programming-language-every-year/
[2] http://www.paulgraham.com/lisp.html
[3] http://www.digitalmars.com/d/
[4] http://www.haskell.org/
[5] http://en.wikipedia.org/wiki/Erlang_programming_language
[6] http://msdn2.microsoft.com/en-us/library/ms364047(vs.80).aspx
[7] http://developer.apple.com/leopard/overview/tools.html

Argentina Has Its Own Linux Distro

Adrian Kosmaczewski

2007-01-31

And it's called UTUTO[1]:

> UTUTO es una distribución de GNU/Linux, denominada en referencia a una lagartilla o Geco así conocido en el norte de Argentina
>
> Su primera versión, grabada masivamente por primera vez en octubre del año 2000 en Argentina por Diego Saravia de la Universidad Nacional de Salta, era muy simple de utilizar y funcionaba desde CD-ROM sin necesidad de instalación. Fue una de las primeras lives del planeta. ISBN 987-9381-06-8.
>
> Declarado de Interes Nacional por la Honorable Camara de Diputados de la Nación Argentina

[1] https://www.ututo.org/

Migration: the Return

Adrian Kosmaczewski

2007-02-01

Yup, the Migration[1] continues. More and more developers are leaving Microsoft technologies behind, and exploring new grounds.

For example, it's interesting to read how Kevin Hoffman (writer at The .NET Addict's Blog[2]) takes his first steps in Cocoa[3]; I've been down that road too, and his impressions are almost the same I had 4 years ago:

> I tried to create the stereotypical "Hello World" application. I dragged a button onto a form and then I figured, "What the hell, this ought to work", so I double-clicked the button and was ready to try and figure out how to get a message box to come up. Much to my chagrin, all I did was pull up the properties inspector for the button.
>
> My first reaction was to drop Xcode and go back to Visual Studio where I felt comfortable. You know how, even though something might actually smell really, really bad, the fact that you're familiar with it and might even call it home makes it endearing? The same is true of a development tool. Regardless of whether your tool is good or bad, the fact that its the tool you've been using for years makes it feel comfortable and familiar, like a security blanket.
>
> I took a step back and tried to figure out what the hell was going on. After a few minutes, I realized that Xcode was actually doing a really, really, really good thing. The vast majority of problems that arise from a poor separation of concerns between the GUI and the underlying code, model, and controller (if you even have such constructs in your app!) stem from the fact that you can double-click a button and immediately start writing code without thinking about the consequences of such a thing.

But even more groundbreaking is to read that Mike Gunderloy[4] (of

[1] /blog/migration/

[2] http://dotnetaddict.dotnetdevelopersjournal.com/

[3] http://dotnetaddict.dotnetdevelopersjournal.com/cocoaday1.htm

[4] http://afreshcup.com/

"Coder to Developer"[5] fame) is also leaving behind Microsoft technologies, testing other grounds[6] and sharing his experiences along the way:

> The last time I completely walked off a job and started over with a new career was around 1992, when I shut down the publishing business I'd built around FACTSHEET FIVE. After a while I ended up writing software, and writing about software, for a living. I've spent the bulk of the last fifteen years developing some amount of reputation and expertise in the Microsoft universe, having published dozens of books and hundreds of articles, worked as an editor and consultant, written (as a subcontractor) parts of various Microsoft products, and so on. I'm also the editor of the Larkware site, which tracks news in the Microsoft software world for developers.
>
> Unfortunately, over that time I've also come to the conclusion that, even though it is staffed largely by smart and ethical people, Microsoft itself represents a grave threat to the future of software development through its increasing inclination to stifle competition through legal shenanigans. Its recent attempt to claim that no one can implement a user interface that looks anything like the Office 2007 ribbon without licensing some nebulous piece of intellectual property represents a new low in this regard.
>
> I'm in a bit of a bind. Unlike fifteen years ago, I've got a family, including four kids, and I can't afford to just walk out on a career that brings in good money. But I rather desperately want to find an alternative. This blog will record some of my explorations as I hunt around in other corners of the software world, trying to decide if there's a viable business plan for me that can include weaning myself off of Microsoft software.

This is a great moment in computing history, I think.

[5] http://www.joelonsoftware.com/articles/CoderToDeveloper.html
[6] http://afreshcup.com/2006/12/9/what-s-going-on-here

Feeling Like a Grown Up Kid

Adrian Kosmaczewski

2007-02-18

Jedi Knights, when they achieve a certain level of maturity in their craft, are told to build their own lightsaber, as one of the final steps of their training. Software developers have similar feats to accomplish at least once in their coding life, like building their own Linux kernel, writing a rant about Microsoft Windows in their blog, submitting a patch for some open-source system, or even starting and managing their own company or open-source project.

Today I've compiled my own kernel from scratch, thanks to the instructions given in the Ubuntu forums. And I feel like I've done another nice, big, useful step in my career. I can only thank all of those who write how-tos and instructions in forums, which is one of the factors that make Linux so great; you can find information about how to do pretty much anything, with incredible levels of detail. I hope that my small contributions might be useful to others! This, for me, is a way to give back to the community as well.

Craving to Read, Back to Commuting?

Adrian Kosmaczewski

2007-02-19

I've got a couple of books on my desk that I'm craving to read! The problem is, lately I'm lacking the time to sit quietly and enjoy them:

- Founders at Work[1], by Jessica Livingston
- The Best Software Writing I[2], selected and introduced by Joel Spolsky
- Eric Sink on the Business of Software[3]
- Options, Futures and Others Derivatives[4], by John C. Hull
- Financial Instrument Pricing using C++[5], by Daniel J. Duffy

One of the good things (maybe the only one) of commuting from Lausanne to Geneva every day back and forth, was the delightful hour of peaceful reading that you get. In the 4 years that I've done that, I've read more books than in any other time of my life.

[1] http://www.foundersatwork.com/
[2] http://www.joelonsoftware.com/articles/BestSoftwareWriting.html
[3] http://www.amazon.com/Eric-Business-Software-Experts-Voice/dp/1590596234
[4] http://www.amazon.com/Options-Futures-Other-Derivatives-John/dp/8120328795
[5] http://www.wiley.com/WileyCDA/WileyTitle/productCd-0470855096.html

El Ruidito

Adrian Kosmaczewski

2007-02-22

En varias ciudades de Suiza, cuando te subis a un trolebus de los viejos, siempre escucharas un “tic-tic-tic-tic-tic…” mientras se mueve el vehiculo. Como si fuese una bomba. El ruidito es escucha sobre todo si te sentas en la parte de adelante, cerca del chofer. Pasaron años hasta que alguien me explico lo que era: los choferes de las empresas de transporte publico tienen que entregar, cada dia, un informe que indica el kilometraje que hicieron, como prueba de que se hicieron correctamente todos los trayectos previstos. A partir de ahi se evalua el rendimiento y los salarios de cada uno.

El ruidito corresponde a un sistema, dentro de una especie de “caja negra”, que llena automaticamente una ficha a medida que circula el vehiculo, y que constituye el informe en cuestion. No se bien que contiene, pero me comentaron que informa sobre la velocidad del vehiculo en todo momento, sobre la distancia recorrida, sobre las paradas efectuadas, los retrasos eventuales, en fin, todo lo que concierne el recorrido del trolebus. Todo.

Es conocido el cariño que le tienen los suizos al control de los empleados. En una empresa donde laburaba hace 2 años, tenia que rellenar 3 planillas distintas de horarios. Asi fue como cultive un odio horrible a SAP. Alguna vez escribi sobre eso. Este control explica el porque del andar panfilo de los autobuses (nunca pretendas ir rapido en un autobus suizo) y los horarios tan precisos que te ponen, no solo en cada parada sino incluso en la web. Y explica tambien ese sentimiento de paranoia que es omnipresente en este pais.

Pero como hacen en Buenos Aires? Es conocido que los bondis a veces circulan a toda velocidad porque les hacen problema si llegan demasiado tarde o demasiado temprano (lo cual, en Buenos Aires, teniendo en cuenta el transito que hay, es una verdadera loteria la mayoria de las veces).

Creo yo, la cosa es asi: en Buenos Aires, cada linea corresponde a una empresa diferente. En Suiza no; todas las lineas de cada ciudad, pertenecen a una empresa unica, estatal, que posee los vehiculos y un equipo de choferes. La diferencia en el control de los choferes estriba en el hecho de que, a diferencia de Buenos Aires, donde las estaciones terminales de las lineas suelen tener un deposito de vehiculos, en Suiza las terminales de las lineas no corresponden usual-

mente con los depositos de los vehiculos; es decir que los patrones no estan en las estaciones terminales verificando si el chofer llego o no. Con lo cual el control se hace de manera automatica, transparente. Si alguien tiene una teoria mejor, adelante!

Tambien es tipico ver como los choferes cambian de turno en medio de los trayectos; es asi como uno, que esta apurado, tiene que presenciar como los choferes cambian de puesto, en medio del trayecto, bajandose del trolebus, saludandose mutuamente durante unos buenos minutos, y despues acomodandose panfilamente en el puesto del conductor.

En los vehiculos mas recientes, empero, el ruidito no se escucha mas. Ahora hay una computadora que transmite toda la informacion en tiempo real a la terminal. Es mas moderno, barato, efectivo, y podes controlar mejor a tus empleados. Creo que hasta transmite el peso del chofer antes y despues del almuerzo.

How to create a BibTeX file from a Delicious Library database

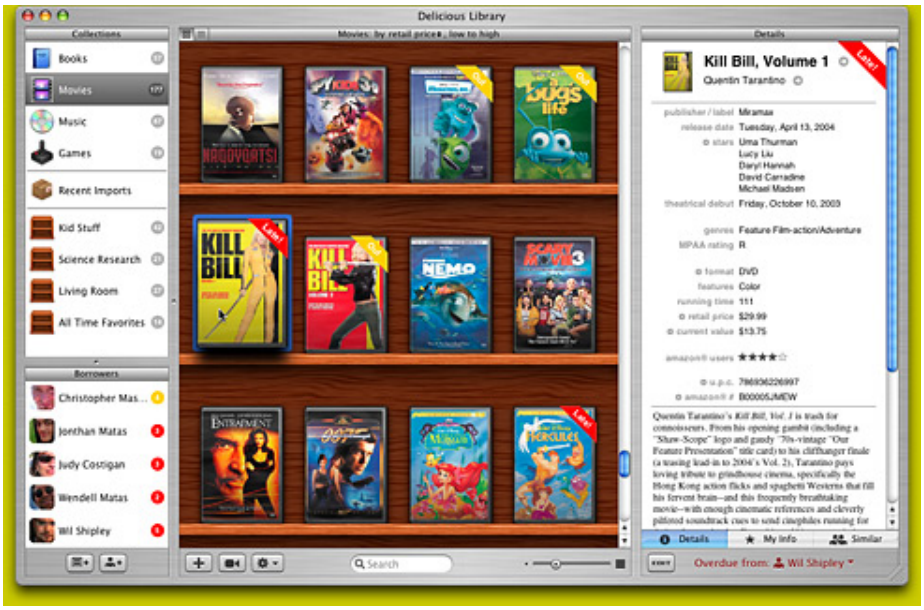
Adrian Kosmaczewski

2007-03-04

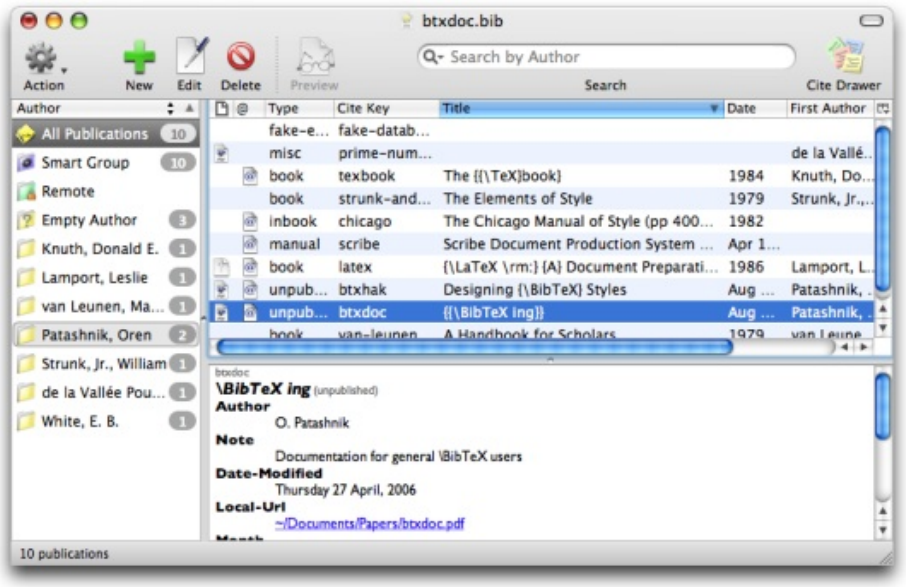
Well, you're pretty much on your own for that :)

I think that Delicious Library[1] should definitely include this export option; it is a great application to manage your books, CDs and DVDs (and even games!). On the other side, I use a lot LaTeX for my Master degree's papers, and being able to export my own library to a BibTeX file is something that really helps.

In the meantime, I created a Ruby script[2] that reads your Delicious Library XML file, and outputs a list of BibTeX entries. Once exported, you can use BibDesk[3] or TeXShop[4] to open and edit the file, and use it for your papers and reports (as I do).



[1] http://www.delicious-monster.com/
[2] bibtexrb.zip
[3] http://bibdesk.sourceforge.net/
[4] http://www.uoregon.edu/~koch/texshop/



Here's a sample BibTeX file[5] with some classics in my own Library.

Hope that you find it useful! Feel free to use it and, as usual, I'm not responsible of what might go wrong with it!

[5] adrianbib.zip

Chupate Esta Naranja

Adrian Kosmaczewski

2007-03-05

el fundador de la “free software foundation”, el carismatico richard stallman, inventor de la licencia GPL, siempre usaba la expresion “free as in free speech, not as in free beer” cuando explicaba el concepto del open source. basicamente, lo que el dice es que “free software” no es “gratis” sino “libre”, pero los anglofonos usan la misma palabra “free” para ambos conceptos.

despues de que se canso de explicar lo de la cerveza, a unos tipos que lo miraban les dio sed, y se inventaron una cerveza libre, que podes copiar libremente; la receta esta disponible online, y esta ofrecida al mundo con una licencia “creative commons”:

[1]

[1] http://www.freebeer.org/blog/

tambien hay una “OpenCola”, cuya receta es libre, y se puede copiar, y tiene una licencia “GPL”:

http://en.wikipedia.org/wiki/OpenCola

por que GPL? porque la GPL estipula que todo producto derivado o que utiliza software GPL tiene que ser ofrecido al mundo con codigo abierto, y ser a su vez licenciado con GPL.

esa clausula de la GPL se denomina “viral”, ya que todo lo que se haga GPL viene a ser GPL automaticamente.

y la freebeer y la opencola son asi. te bajas las recetas y las producis. les pones tu marca. las vendes si se te canta el firulete. ahora bien, estas obligado a compartir la receta (aunque la hayas modificado) y a hacer referencia al origen de tu bebida.

asi van las cosas por ahi. salud!

blogalité, ilegalité, fascistité

Adrian Kosmaczewski

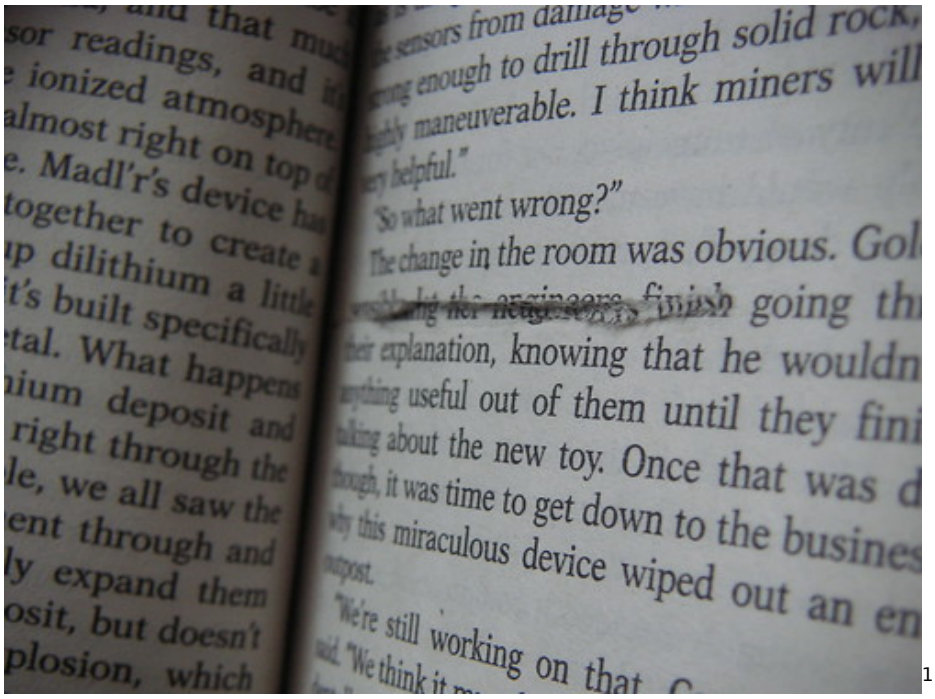
2007-03-08

http://www.odebi.org/

te habras enterado que en francia hay elecciones. basicamente hay tres candidatos que pueden ganar: nicolas sarkozy, segolene royal (que podria llegar a ser la primer mujer presidente de francia) y francois bayrou. sarkozy es el de “derecha” y es ministro del interior actualmente. la segolene es de “izquierda”. bayrou es del “centro”. y yo vengo de vicente lopez.

este sarkozy acaba de hacer pasar una ley en francia que hace ilegal (agarrate) la publicacion de informaciones sobre actos de violencia publicos, por parte de ciudadanos que no son periodistas acreditados. en otras palabras, **el blogueo de denuncia es a partir de hoy ilegal en francia.** ponele que ves a la cana golpeando un tipo, como en el caso de rodney king, el tipo que fue filmado mientras unos canas lo golpeaban en california, alla por el 91. el tipo que filmo eso lo difundio, le hicieron juicio a los canas, y como quedaron libres, los angeles paso por sangre y fuego en el 92.

bueno, eso de difundir un video asi es ilegal en francia hoy. tambien lo es cualquier difusion no autorizada de informacion relevante a la vida publica francesa. un blog como enta seria ilegal ipso facto en argentina si algo asi entrase en vigor.


[1]

la excusa de sarkozy es que hace unos dias, unos pibes publicaron en youtube.com un video de como cagaban a palos a otro, filmado con el celular. supuestamente es para evitar eso.

la ley paso con gran silencio, sin mayor eco en la prensa. para mi, eso es como el incendio del reichstag, o como el derrumbe del world trade center, es un acto de violencia, pero hecho con toda la fineza e hipocresia que solo los franceses pueden lograr.

enfin, ya te imaginaras que clase de personaje es sarkozy. es un tipo muy peligroso, y obviamente esta por ganar las elecciones, actualmente siendo primero en las encuestas (aunque no por un gran margen).

eso.

pd: leido en ohmynews.com:

> Were they aware of it, the majority of French parents would approve of this new law because it was presented in parliament as a solution to the "happy slapping" problem. Those infamous videos of youngsters beating each other up, sent over mobile phones, led to shocking incidents involving children in France. The general lack of reaction can also be explained by the fact that this particular law was bundled in a set of laws designed to "fight juvenile crime." What is raising concern among teachers, social workers and doctors, right now, is another aspect of this law package: their new

[1] https://www.flickr.com/photos/zerok/19848092/

obligation to hand over confidential information on juvenile offenders to the police or mayors. The video ban slipped by unnoticed.

(…)

Their real concern – and mine – is the growing trend under this government to enact clumsy, vague and potentially dangerous laws under noble disguises such as the "fight against violence" or to "protect children from pornography." RSF is equally worried by another law proposal, designed this time to "protect youth." If it passes, a national committee (chosen by whom?) will be allowed to grant "quality stars" to information Web sites based on their "ethics!" What exactly is "quality" in information?

Herejía

Adrian Kosmaczewski

2007-03-08

Y quitarle al dios de los cristianos
su corona de espinas
“Jugar por Jugar”, Joaquín Sabina

Soudain toute la ville s’arrête
il paraît que les fleuves ont grossi!
C’est Dieu qui s’est assis sur le rebord du monde
et qui pleure de le voir tel qu’il est
“Assis sur le rebord du monde”, Francis Cabrel

Lo que se me ocurre ahora, en otras epocas hubiese sido suficiente para que la Inquisicion me mande a la hoguera.

Y pensandolo bien, en estas epocas tambien.

A ver, empecemos:

1) “Padre nuestro que estas en los cielos”.

Personalmente, considero que esta en todos lados, no solamente en el cielo. La distincion entre cielo y tierra (ver mas adelante) no me convence demasiado. Considero al mundo como una unidad, un poco como el concepto esoterico sobre la astrologia, según el cual, “lo que es arriba, es abajo”.

Sigamos.

2) “Santificado sea tu nombre”.

Acaso no lo es por definicion? Y este deseo, a quien va dirigido? Quien santifica el nombre de Dios sino Dios mismo? Si santificar es consagrar, elevar, transformar una nocion en algo supraterrenal; y si Dios nos impone como condicion sine qua non de su gracia la creencia UNICA de su existencia, quien mas que Dios para santificarse a si mismo? Que tenemos entonces nosotros que decirle? Quienes somos para decirle eso???

3) “Venga a nosotros tu reino”.

Aquí el termino REINO no es el mejor: el concepto politico de la palabra tergiverso todas las interpretaciones de la frase. Frase que es por demas imperativa, como tantas otras del Padrenuestro, lo cual es

(según mi opinion) un atentado contra los votos de humildad que son supuestamente la calidad primera de la cristiandad.

Queda por saber que es el Reino, en definitiva. Dejo eso a la Biblia, en la cual los apostoles escribieron algo sobre el tema (sobre todo en el libro del apocalipsis).

4) “Hagase tu voluntad, asi en la Tierra como en el Cielo”.

No comments. Veanse los comentarios del punto 2, con algunos agregados del punto 1.

Otra vez el tono imperativo, la dicotomia Tierra/Cielo, la soberbia de creer que nuestra opinion puede ser trascendente para la entidad responsable de la existencia misma del todo.

5) “El pan nuestro de cada dia danoslo hoy…”.

Nos lo merecemos? Lo hemos buscado? Hemos hecho algo tan trascendental que nos permita darnos el lujo de EXIGIR (porque ese es el tono de la frase) un sustento alguno? Por que no pedirle mas serenamente que nos ayude a conocernos en mayor profundidad, para asi corregir nosotros mismos nuestros defectos â†’ y ayudarnos a cambiar de perspectiva para que nuestra subjetividad no nos deje ciegos???

No. Se prefiere una postura mas acomodaticia, una solicitud y la espera pasiva sin ninguna autoexigencia.

6) “… y perdonanos nuestras deudas, asi como nosotros perdonamos a nuestros deudores”.

Para que contraer deudas de las que tengas que pedir perdon a Dios??? Por un lado, la Iglesia se cansa de decir que Dios es amor y perdon infinitos; por otro lado, es la frase a la que la humanidad da menos atencion en toda la oracion. Sinceramente conozco pocos que perdonen deudas, de cualquier indole.

7) “Y no nos dejes caer en la tentacion…”. (porque somos tan tarados que hace 2000 años que no podemos dejar de hacerlo, al menos; nos hiciste bastante flojos, los monjes budistas son la excepcion â€“ pero no rezan padrenuestros… tendra algo que ver???)

8) “…mas libranos del mal”.

Aquí mas = pero. No corresponde otra conjuncion aquí, a saber y???

Tal y como esta construida, la frase deja suponer varias cosas:

a) Es probable que Dios te largue en banda, lo cual suena mezquino (luego improbable) de una fuente infinita de amor. Contradiccion, pues.

b) Dios no hace todo a la vez: o te impide caer en la tentacion, o te libra del mal. En la frase se hace mayor hincapie en que te libre del mal, por lo que se deduce que si no te protege de caer en la

tentacion, al menos te librara del mal, luego la tentacion no sera tan mala, o estara incluso exenta de mal. El resto se los dejo a ustedes.

9) “Porque tuyo es el Reino, el Poder y la Gloria, por todos los Siglos (o por los Siglos de los Siglos)”.

Este es un agregado de la Iglesia Protestante, que es inexistente en el Padrenuestro canonico de la Iglesia Catolica Apostolica Romana.

Varias observaciones:

a) La nocion protestante de la propiedad privada aparece aquí en su mas fuerte expresion; tal y como la ausencia para los catolicos romanos, que (aunque no lo digan) subordinan todo al Vaticano (inexistente, no olvidemos, en la epoca de Cristo).

b) Dios ya sabe que todo eso es suyo, no creo que necesitemos recordarselo.

c) Otra vez, aparece el Reino, que es suyo, sobre el cual le pedimos anteriormente que venga a nosotros, en prestamo, generando una deuda, que luego le pedimos que nos perdone, y del cual mas tarde le reconocemos la titularidad. Uf! O sea, una estafa de dimensiones divinas.

d) El Poder, obviamente, es de Dios (despues de todo, El creo el Universo, lo cual no es poca cosa) pero... la Gloria? Gloria de que? De ver el mundo como esta, hay algo para vanagloriarse???

10) “Amen”.

“Asi sea”, en hebreo. Lejano recuerdo de que Jesus fue judio, aunque a muchos dirigentes de extrema derecha no les guste la idea. Una manera de concluir una plegaria tipica de las grandes religiones monoteistas occidentales â€“ que, vale la pena recordarlo, descienden todas del mismo tronco historico (Abraham)

Decia entonces, es una manera de concluir la plegaria un tanto sufrida; con la conviccion de que la intervencion propia en el comportamiento divino es infima, improbable, lejana, inaudible. Casi inutil.

Finalmente, cuantas veces rezamos sin pensar en esto? Pensamos que Dios esta alla, sentado en su trono, en el trono de su Reino?

Quiero creer que Dios esta en mi corazon, es mas, quiero creer que soy la encarnacion local de Dios, que Dios llora conmigo, rie conmigo, aprende conmigo. Conmigo, en mi y a traves mio, asi como de todos nosotros.

No hay tal distancia entre Cielo y Tierra; la muerte no es el fin sino un nuevo comienzo. Tal y como creo fervientemente que Jesus existio y quiso hacer mejor el mundo en el que vivia, tambien creo en que los apostoles escribieron muchas boludeces. Entre ellas transcribieron para el carajo una oracion, que fue manipulada durante 2000

años para servir a una extraordinaria y nefasta manipulacion de la mente humana. Y que dio lugar a uno de los aparatos de estado mas violentos y organizados que haya conocido la humanidad.

Ya no se cuentan los crimenes cometidos en nombre de Dios.

Es hora de rezar de otra manera. Dios se debe haber ido a otra galaxia, ya debe estar podrido de tanta hipocresia.

Preferred Programming Languages

Adrian Kosmaczewski

2007-03-09

There are basically 5 languages that I really like. For several reasons. 2 of them are proprietary, while 3 are open-source. 2 are statically typed, and 3 are dynamically typed. All are fully object-oriented. 3 can be used for web development, 4 for desktop apps. And none is Java.[1]

My preferred compiled language? **C++**. You get the full-power of many operating systems, from the same source code, without compromises; OK, it takes time to get used to the lack of memory management, but it is highly portable, extremely fast, and there are a couple of extremely good libraries around the corner that fully justify getting into it: POCO[2], Juce[3], Boost[4], wxWidgets[5], Qt[6], CppUnit[7], Doxygen[8]... Once you get some habits and good practices you can very quickly put some complex code together. What are the best **books** you can read about it? Stroustrup[9]'s to start, Meyer[10]'s to go deeper, and O'Reilly[11]'s to go faster.

My preferred scripting language? **Ruby**. As someone said, "elegant as Smalltalk, simple as Python, and pragmatic as Perl". A pleasure to use; you can very quickly create complex applications in just a couple of lines of code. And with Rails[12], well, I got the impression that finally someone got web development right. It's got the best of PHP, the best of WebObjects, the best of Ruby.

My preferred web language? **JavaScript**. It might be the most misunderstood programming language[13], it is one of the nicest too. Dynamic, extensible, flexible as rubber. OK, not the fastest, but does it

[1] /blog/not-exactly-what-i-meant/
[2] http://www.appinf.com/poco/info/index.html
[3] http://www.rawmaterialsoftware.com/juce/
[4] http://www.boost.org/
[5] http://www.wxwidgets.org/
[6] http://www.trolltech.com/products/qt/
[7] http://cppunit.sourceforge.net/
[8] http://www.stack.nl/~dimitri/doxygen/
[9] http://www.research.att.com/~bs/3rd.html
[10] http://www.awprofessional.com/bookstore/product.asp?isbn=0321334876&rl=1
[11] http://www.oreilly.com/catalog/cplusplusckbk/
[12] http://rubyonrails.org/
[13] http://www.crockford.com/javascript/javascript.html

always matter?

My preferred Apple language? **Objective-C**. A mix between Smalltalk and C? A static scripting language? Or a dynamic language with a compiler? Objective-C is OOP as it should have always been done. And Cocoa[14] is in my opinion the most comprehensive, extensible, advanced and stable foundation ever created for application development.

My preferred Microsoft language? **C#**. Even if I left my Microsoftie life behind, I think that .NET and C# are probably the best products ever done by this company. Ever. .NET is a strong and complete foundation, and it does not surprise me that companies choose it for their Windows-based development. There's simply no better way to do Windows development. I said Windows: for web development, I think that ASP.NET sucks. Rails rules.

[14] http://developer.apple.com/cocoa/

Truquito CSS

Adrian Kosmaczewski

2007-03-12

...que por ahi te sirve. tiene que ver con la presentacion de las paginas de un sitio cuando se imprimen.

constato:

- que uno no quiere que la gente, cuando imprime paginas de tu sitio, imprima los menues que aparecen a los costados, u otras partes de la pagina, porque ocupan espacio en la hoja y no sirven de nada (no se pueden clickear!);
- queres que los links que aparecen subrayados en pantalla, no aparezcan subrayados en la pagina impresa, y que ademas aparezca, entre parentesis, la direccion web a la que apuntan (lo cual es interesante para referencia futura)
- que no queres tener que mantener una “version imprimible” o “printer friendly” de la pagina, que es considerada una mala costumbre, y te duplica el quilombo de tu sitio
- queres usar algo estandar

entonces, en el CSS de tu blog, pone algo asi:

```
@media print {
    a {
        color: black;
        text-decoration: none;
    }

    .noprint {
        display: none;
    }

    a:after {
        content: " (" attr(href) ") ";
    }
}
```

explicacion:

- `@media` delimita un sector de tu CSS para que se use solamente... en algun tipo de medio: print, screen y all son los valores mas comunes.

- `a:after` “inyecta” despues de cada elemento “a”, un texto; en este caso, el valor del atributo href.
- si le pones `class="noprint"` a cualquier elemento, este desaparece de la version impresa.

lamentablemente esto no funciona en internet exploder 6, pero si en firefox (supongo que en safari tambien, pero no “testie” aun).

si queres fijate como aparece impresa esta pagina cuando la imprimis con firefox: cada link de la pagina aparece en parentesis en el texto impreso, para que no se pierdan esos datos.

que se yo, me parecio piola. contame que te parece!

Reducing Code Entropy

Adrian Kosmaczewski

2007-03-18

This is a rant: **I am tired of seeing virtual methods implemented in child classes that, at some point or another, call the method of the same name in the base class.** For me this is a sign of poor architecture. A bad, bad smell in code.

Let's say that you have a base class, called, well, `BaseClass` (the examples are in C++, but you can take this idea to any other OO language):

```
class BaseClass
{
public:
    // constructor, destructor, copy
    // constructor, assignment operator...

    virtual void doSomething();
}

void BaseClass::doSomething()
{
    // provide some basic behavior here,
    // and a comment that says something like:

    // IMPORTANT NOTE to implementors of subclasses:
    // this method should always be called before your code!
}
```

And then you derive a class from this base class, called, well, `DerivedClass`:

```
class DerivedClass : public BaseClass
{
public:
    // constructor, destructor, copy
    // constructor, assignment operator...

    virtual void doSomething();
}

void DerivedClass::doSomething()
```

```
{
    BaseClass::doSomething();     // WTF???

    // do something else...
}
```

As stupid as it might sound, this code is hard to understand, debug and maintain, because it includes an unneeded dependency from the child to the base class - that is, another one, besides the fact that one derives from the other!

The problem is that the `doSomething()` message appears here as actually **two messages**: one which is mandatory and generic (like initialization or verification), and the other, a specialized one (which is why people use polymorphism, actually). One is public, the other is not. Mixing both messages creates a semantic problem in the above code, which ultimately makes the code harder to understand.

This might lead a myriad of other small things; what if another developer (just landed in the team), months later has to add a new child class of `BaseClass`, and forgots to (or just does not know that she should) call the `BaseClass::doSomething()` method? This might cause pain, either because it generates a bug that's hard to track, either because the developer spends time trying to understand why the new class does not work while she's working on it. In any case, it is a cost that you, as an architect, might have avoided in the first place. And if your designs are poorly documented (which is a sad truth), or if the developer that created the code has gone away from the team (which happens a heckuva lot of times every day), well, your chances of losing time and effort in your project will grow accordingly.

In my opinion, such an approach goes against the very idea of polymorphism, and I prefer to refactor such a code this way:

```
class BaseClass
{
public:
    // constructor, destructor, copy
    // constructor, assignment operator...

    void doSomething();

protected:
    // a pure virtual, abstract method
    // without implementation, working as a
    // "hook" for subclass implementors
    virtual void doSomethingElse() = 0;
}

void BaseClass::doSomething()
{
    // provide some mandatory, basic behavior here
```

```
    // hook for subclasses to extend this behavior!
    doSomethingElse();
}
```

And then:

```
class DerivedClass : public BaseClass
{
public:
    // constructor, destructor, copy
    // constructor, assignment operator...

protected:
    // provide some extended behavior!
    virtual void doSomethingElse();
}

void DerivedClass::doSomethingElse()
{
    // do something else...
}
```

Ahhh... I much more prefer this way of doing things:

- First of all, it makes `BaseClass` an abstract entity, an interface, a pure thought creation, a divine entity above all the earthling vacuum; you want to program against interfaces, and not be tempted to use them as concrete things. That's what subclasses are for.
- The call to `doSomething()` will always, always (let me repeat that, **always**) be called, no matter how sloppy the creator of `DerivedClass` is. The worst thing you can have is an empty `DerivedClass::doSomethingElse()` implementation. In that case, no harm.
- If the developer is so, so, so sloppy that it forgets to implement `DerivedClass::doSomethingElse()`, well, guess what: at some point in time, either the compiler or the linker will complain! And this is a **good thing** (at least in languages that feature a compiler, of course!)
- You've unknowingly created a nice API, by the way. You can document this with a good UML diagram, and it will stand out in your Doxygen or JavaDoc documentation.
- It's easy to understand and hard to break, as all good designs should be.
- Your developers will love your work, and your karma gets a bonus point. The next life you might as well approach a higher level of consciousness and save Mankind from its horrible destiny. (OK, maybe this is too much, but hey, who knows?)
- The approach is called the "Template Design Pattern" and this will add another bonus, but this time for your CV ;)

Hope this helps!

TCP Friendliness

Adrian Kosmaczewski

2007-03-20

TCP congestion control tries to bring solutions to the problem of packet loss, which happens in networks for several reasons:

- Increasing number of hosts connected to the network;
- Increased traffic to and from the hosts;
- Limited capacity of routers in between hosts and networks (Kurose & Ross, page 254).

Congestion is an important problem raised by multimedia streaming applications, where a high quantity of UDP packets (typically with audio and video information) is sent through the network with important rates. The problem with this approach is that UDP has no congestion traffic, and as such packets might be dropped by routers if a buffer overflow occurs. TCP, on the other hand, provides network congestion management, but

> TCP is not well-suited for many applications such as streaming multimedia, because TCP congestion control algorithms introduce large variations in the congestion window size (and corresponding large variations in the sending rate)

(Kyasanur)

Network congestion, in turn, has complex consequences in terms of lower perceived quality of service, which is easily seen by the low quality of IP telephony communications in peak hours, bad quality video connections, and the commercial implications of these problems (difficulty to newcomers to provide innovative services, low acceptance by end users, etc).

The ideal solution to the problem of network congestion would be to be able to benefit from the congestion management features of TCP, while being able to deliver complex information at high rates, coupled with the low overhead of UDP. **“TCP-friendly”** is the generic name given to a set of algorithms that provide the advantages of congestion control while using UDP as the transport protocol, and is currently a hot research topic (this is easily seen by the high number of papers that are returned as search results for the terms “TCP-friendly”: http://www.google.com/search?q=tcp%20friendly).

The IETF has published a standard “TCP-friendly” protocol, called TFRC

or “TCP Friendly Rate Control”. The official specification states that

> TFRC is a congestion control mechanism for unicast flows operating in a best-effort Internet environment. It is reasonably fair when competing for bandwidth with TCP flows, but has a much lower variation of throughput over time compared with TCP, making it more suitable for applications such as telephony or streaming media where a relatively smooth sending rate is of importance.

(Network Working Group, 2003)

The issue of all current research in the subject will most likely have a huge impact on the Internet, with important consequences in terms of quality of service, reliability and speed, and also in terms of commercial and business implications.

[1]

References

Kurose, J. F.; Ross, K. W.; “Computer Networking, Third Edition”, Addison Wesley, ISBN 0-321-22735-2

Kyasanur, Pradeep; “Analysis of TCP-friendly Congestion Control Algorithms”, http://www.crhc.uiuc.edu/~kyasanur/papers/cs497rhk_tcp.pdf (Accessed December 10th, 2006)

Network Working Group, “RFC 3448 - TCP Friendly Rate Control (TFRC): Protocol Specification”, January 2003, [Internet] http://www.faqs.org

[1] https://www.flickr.com/photos/bopuc/405855225/

/rfcs/rfc3448.html (Accessed December 10th, 2006)

Google Everywhere

Adrian Kosmaczewski

2007-03-21

It all started with the search engine. I think it was sometime back in 2000, while I was working in Argentina.

I was an avid everyday AltaVista user; I found it quite useful since 1996 (when I started browsing using Netscape at the University of Geneva) and I was frankly happy with it; I was not looking for another search engine. All the others I had used were not as good, so I kept using AltaVista for years. I knew how to cope with the poor result sorting, and I would usually find what I was looking for, typically as a link buried in page 4 or 5 of the search results screen. It was like that, it took time, and more often than not, I did not find what I wanted.

However, one day I saw a colleague at work looking for something at a new search engine, called “Google”. I asked her the URL, typed it on my browser, and started to use it. I haven’t stopped since. Google was (and is) amazing. It is rather unusual for me to have to go to page 2 or 3 of the search results: I usually find what I’m looking for, right in the first page. The “I’m Feeling Lucky” button is the sign that shows how smart the backend engine is (as well as the team that created it). Joel Spolsky was among the first to notice this at the time, I think.

[1]

I used Google just like that, for a couple of years, and in 2002 I discovered the Google Toolbar. That changed a lot of things in my day-to-day workflow: it seemed stupid, but having that search box in the browser made everything so much faster! I started to include it in every browser of every computer I used, either at home or at work. But then again, I was not using Google for anything else than search. I knew that they offered other services, but I wasn't too much interested in looking at them. What I learnt to use was the Google search syntax; I found that using "site:" and "filetype:" in my queries were simple yet powerful ways to find what I was looking for.

The IPO in 2004 made the big news. Everyone started to talk again about the web, like when Yahoo! and Netscape went public, nearly 10 years earlier. The big NASDAQ boom seemed to be so far away, and so close at the same time.

Finally, in February 2005 I was invited to open a Gmail account. And I think that everything changed that day; Gmail was so advanced (and still is!). I've used Hotmail since July 1996 (I am sure to have signed up for one of the first Hotmail accounts), and I've also used Yahoo! Mail extensively; but Gmail was years light ahead of everything. At that time, AJAX was appearing as the new hot technology, and again, Joel saw Google's advance in Google Suggest.

Nowadays, I'm at Google nearly all day long:

- I get informed on what happens elsewhere thanks to Google Reader, the greatest RSS reader I've ever seen;
- I use Google Docs & Spreadsheets extensively for my own documents, particularly when I have to cooperate with other people

[1] http://web.archive.org/web/19990422191353/http://google.com/

remotely on the same files;
- I use the latest version of the Google Toolbar in my Firefox browsers, with nearly all of the cool features turned on; for example:
  - I open Word and Excel files using Google Docs & Spreadsheets;
  - I make it handle "mailto:" links with Gmail;
  - I added the Google Reader button, to see if new information is available;
- I keep an eye on my blog thanks to Google Analytics and the Google Webmaster Tools;
- I have a personalized Google News page, with some keywords that I follow closely every day;
- All of my bookmarks are now online, hosted in Google Bookmarks; this is the first time in more than 10 years that I have found a useful way to centralize all of my bookmarks and "favorites" in only one, single, accessible location (I do not use Google Browser Sync at all, though);
- My personal agenda is hosted in Google Calendar;
- I host a couple of programming projects in Google Code: DVDRental and JuicyCRM;
- I use Google Talk with some of my friends, particularly if they are connected simultaneously to Gmail (that integration is awesome);
- I use several of Google's services for preparing towards my Master's degree in IT:
  - The most important, is definitely Google Notebook, together with Firefox Google Notebook Plugin: they both allow me to put together ideas and references for new papers, and then I can export all the links and references in Google Notebook to a new document in Google Docs & Spreadsheets...;
  - I use Google Scholar a lot, to get links to PDF papers and reports;
  - Google Groups, with an amazing mass of newsgroup postings, makes a really good reference, particularly for Computer Science subjects;
- My Search History helps finding links that "I knew I saw" sometime in the past, and you cannot retrieve otherwise;
- I host some photos in my Picasa account (though I prefer Flickr, actually);
- When using Windows XP comptuters, Google Desktop makes it easy to find information on the hard drive (but I do not use any "widgets", I prefer to keep my desktop as clean as possible);
- I've even played a little with Google Pages... actually I've even written a blog entry about it[2].
- The other day I've discovered Google Base... so one of these days I'll post something about it as well. :)

Just to tell you, even in my job toilets I've put some of the Google

[2] /blog/do-it-yourself-now-and-then/

Testing team “Testing on the Toilet” papers, with great success!

But Google’s advance goes beyond what eyes can see; one of the key concepts in their infrastructure seems to be “MapReduce”, a whole infrastructure that allows them to parallelize complex problems in small chunks, in many different programming languages, using their server cluster to do the tough jobs. Again, Joel has written a very interesting article about it. After having worked for Microsoft, Joel is able to see the differences between Google and his former employer, and his insight is really awesome.

I think that Google makes the web a useful tool. I nowadays spend more time in front of a browser (usually Firefox, or Camino[3] on the Mac) than using any other application. I can access all of my stuff easily, fast, without problems. It looks like “the network is the computer”, after all. Sun was right... but Google delivered it through the web.

[3] http://www.caminobrowser.org/

A Happy Couple

Adrian Kosmaczewski

2007-03-24

My wife Claudia and me (courtesy of South Park Studio)

AOP and the DataServices Project

Adrian Kosmaczewski

2007-03-27

Five years ago I worked as a Software Engineer for a startup, based in Geneva, Switzerland, which had the goal of creating a web-based systems management console, to control and monitor the status of large computer installations, much like Microsoft SMS (Systems Management Server)[1] does. This tool would eventually benefit from being a web-based application, and as such it could be used from anywhere, without having to install a "fat client"; just launch a browser, point to a particular URL, and you are done.

During the project, I was able to work towards the creation of the first AJAX application I've ever seen (this was 2002!), and also, to use Aspect-Oriented Programming techniques for the first time.

The Architecture

This tool was designed as a three-systems project:

- On the server side, the back-end: it used "software spiders" that crawled your local network for information about the domain computers, using their WMI (Windows Management Instrumentation) services[2] to gather useful information such as processor type, network card address, memory size, CPU usage, any kind of information. This information was sent to the server, where it was stored in a SQL Server 2000 database, for quicker retrieval and reporting.
- On the client side, a web based application, running on top of IIS (Internet Information Services) and implemented as an ASP application (Active Server Pages). This tool was the first AJAX (Asynchronous JavaScript and XML)[3] application I had seen at the time (2002!), giving users a real-time experience of what was going on in the network; for example (this was impressive) you could just plug a new computer on the domain, and a couple of seconds later, the information would appear in your web browser automatically, without having to refresh the page (the

[1] http://www.microsoft.com/smserver/
[2] http://msdn.microsoft.com/library/en-us/wmisdk/wmi/wmi_start_page.asp
[3] http://en.wikipedia.org/wiki/AJAX

client AJAX component polled the back-end every 5 or 10 seconds for updates).
- In between both ends, the ASP application connected to the SQL Server backend using a COM+[4] "proxy" component, which was the only means of communication between both subsystems.

The development team was split in three teams, one working on the spider+database backend, and the other (where I was) working in the ASP client application. Finally, a highly-skilled C++ programmer had the task of building the "proxy" component that allowed both systems to communicate. This component was critical, and was designed for extreme high performance. Its design would deserve an article just for itself... just to mention, it featured transactional requests, complex event and feedback models and had extensive queuing capabilities, via MSMQ (Microsoft Messaging Queue).

Just for the record, regarding the methodologies used in the project, it is worth noting that the whole of the project was managed using the Rational Unified Process (RUP)[5]. It must be said that in the case of this startup company, this methodology was an "overkill", and documentation was never kept in sync when changes were done to the architecture. Being a startup company required to be able to adapt the software as quickly as possible, and often this meant skipping the documentation process.

The DataServices Project

The COM+ component had extremely high throughoutput and small response times, but for the sake of interoperability, it required the ASP client application to send and consume complex XML requests and responses, using an industry standard called "CIM" (Computer Information Model). This XML standard, designed by the DTMF (Distributed Task Management Force)[6] is part of the WBEM initiative[7] allowing for XML-based interoperability of computer management systems. It is worth noting that the Microsoft Systems Management Server (SMS) is based on this standard as well, and that the Windows WMI services are also based on these standards.

The problem, for the web applications team, was then to create long and complex XML requests to send to the COM+ proxy component. The web developers, already struggling with a complex AJAX-based UI, had real trouble to figure out the different parameters for these calls, and to create the proper XML structure (The CIM XML format is an extremely complex tree structure, with several layers of information and a high level of redundancy). This led to longer development cycles, since every new use case implementation required a good deal

[4] http://www.microsoft.com/com/
[5] http://www-306.ibm.com/software/awdtools/rup/
[6] http://www.dmtf.org/
[7] http://www.dmtf.org/standards/wbem/

of new requests to be created to the server, with low reuse of already existing requests.

This is how the "DataServices" subproject was born: to create a layer of abstraction between the COM+ component and the dynamic HTML application. The goal was to provide a simpler way to connect the backend with the user interface, so that the development team in the UI layer could concentrate in the user experience, rather than in creating long XML documents. This was dragging development times, and was a well-known bottleneck for the project.

To avoid impacting the current ASP application, and to leverage XML capabilities, this project was created as an ASP.NET application, using the recently released .NET Framework, version 1.0 (released in February 2002) and the C# language.

This ASP.NET application would handle requests from the dynamic HTML application, but providing an API that would be much simpler for developers; for instance, instead of having to create long XML strings to retrieve the list of processes running in a given computer of the domain, the developers could just call "ComputerSystem.GetRunningProcesses(machineName)" ("facade" methods) and that would do it.

Aspect-Oriented Programming

Once the DataServices subproject began growing, we noticed that we were performing the same operations everywhere, for each "facade" method call:

- Deserialization of requests
- Security checks
- Instrumentation (mainly logging, but also performance management and throughoutput)
- Back-end connection management (opening, pooling, closing)
- Error management
- Serialization of responses

The same lines of code were repeated around almost every internal operation done by the DataServices infrastructure, but this code could not be properly encapsulated in an external component, to be reused instead of duplicated.

More or less at the same time, one of the team members read a paper about the newly-born idea of "Aspect Oriented Programming" (AOP)[8], and about the basic AOP capabilities of the .NET Framework (Dharma, Fell and Sells, 2002).

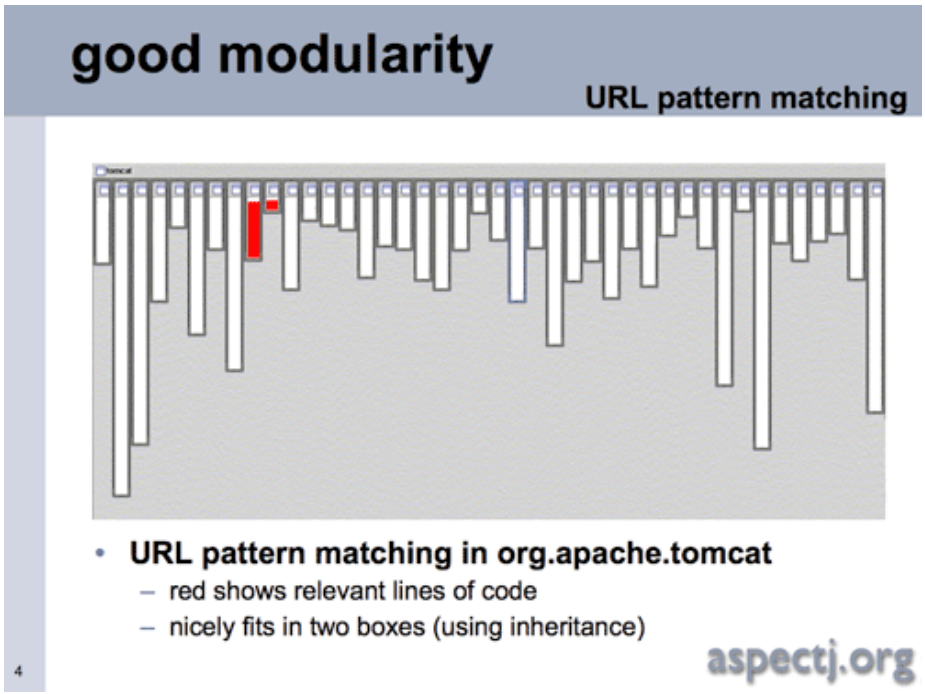
Gregor Kiczales from Xerox PARC, one of the creators of AOP, explains that

[8] http://www.aosd.net/

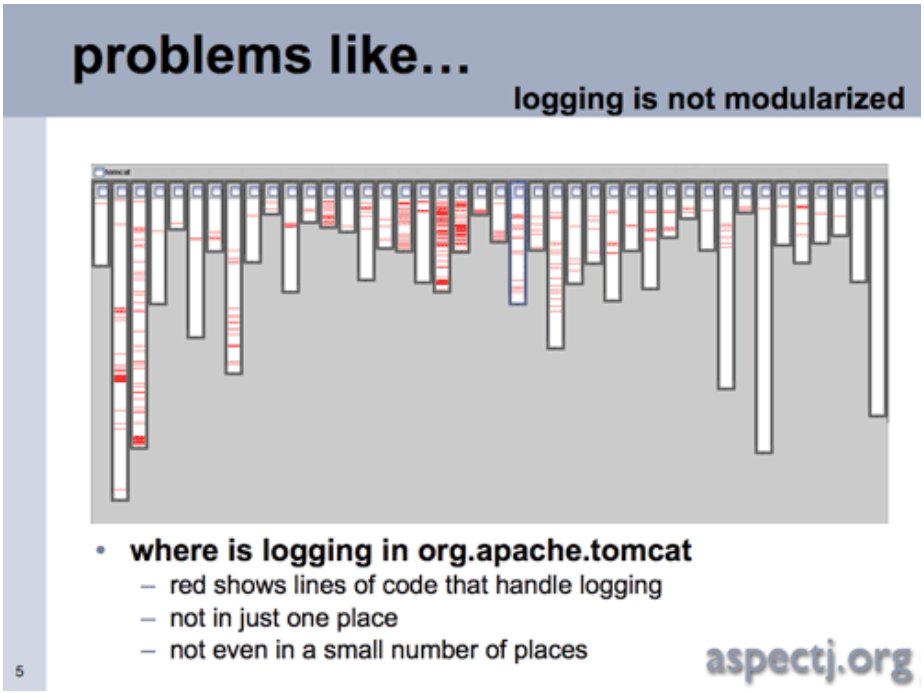
> “We have found many programming problems for which neither procedural nor object-oriented programming techniques are sufficient to clearly capture some of the important design decisions the program must implement. This forces the implementation of those design decisions to be scattered throughout the code, resulting in”tangled” code that is excessively difficult to develop and maintain. We present an analysis of why certain design decisions have been so difficult to clearly capture in actual code. We call the properties these decisions address aspects, and show that the reason they have been hard to capture is that they cross-cut the system’s basic functionality. We present the basis for a new programming technique, called aspect-oriented programming, that makes it possible to clearly express programs involving such aspects, including appropriate isolation, composition and reuse of the aspect code.”

(Kiczales et al., 1997)

The following two images highlight the location, in the Apache Tomcat project, of different sections of code that handle URL pattern matching and logging:



(Source: Hilsdale, Kiczales, 2003)

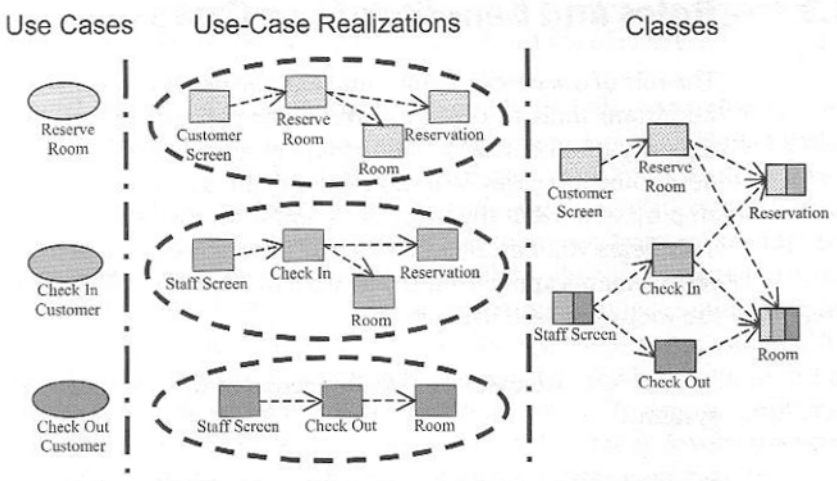


(Source: Hilsdale, Kiczales, 2003)

The idea of AOP is to provide a mechanism to centralize these lines of code in a single location, which would provide better management and maintainability in the future; Jacobson further explains that

> "Even though peer use cases are separate during requirements, their separation is not preserved during implementation. The realization of a use case touches many classes (scattering), and a class contains pieces of several use-case realizations (tangling). As a serious consequence, the realization of each use case gets dissolved in a sea of classes."

(Jacobson, Ng, 2005)



(Source: Jacobson, Ng, 2005, page 33)

Regarding DataServices, suffice to say that AOP seemed to us as a good solution for our problem. Tempted by the discovery of a new paradigm, we decided to give a try to AOP, and the results proved to be worth the effort. In our case, we intercepted each message sent to the classes implementing the use cases (like the ComputerSystem.GetRunningProcesses(machineName) example above), "injecting" behavior before and after the associated method calls, providing the services enumerated above (instrumentation, error handling, etc).

Using DataServices, you could define methods like this:

```
[Log()]
[RequiresRole(Role.Admin)]
[Database(ConnectionType.SQLServer)]
public bool CreateRecord([RegExp("[a-z]*")] string name,
[Range(1, 99)] int age)
{
    // just open the connection and insert,
    // no further checking needed!
}
```

As you can see the `CreateRecord` method contains .NET attributes that define the valid ranges of execution for the parameters, and has some other attributes that define pre- and post-conditions to be checked prior to execution. This allowed us to centrally manage a quite large framework (around 20 classes, or around 100 quite complex methods) and centralize the debugging, security, logging and parameter checking routines into a single location.

Moreover, using configuration files, one was able to "inject" ("weave", using AOP terminology) more behaviors into this mechanism without having to recompile the application (this is known nowadays as "Dependency Injection" or "Inversion of Control" - Fowler, 2004), while

the core “facade” methods only contained specific instructions related to the business rules that they implemented. The code was lighter, easier to read and maintain, and largely self-documenting.

It is worth noting, however, that we used a very small subset of the functionality that AOP provides, that is, the runtime interception of messages sent to objects and weaving aspects before and after method execution:

> “Another way to understand AOP is in terms of how it works. A common misconception associated with this perspective is to equate all of AOP with just one part of the supporting mechanisms. This kind of error is analogous to saying that OOP is just abstract data types. Probably the most common mechanism error is to equate AOP with interceptors.(…) It’s true that AOP does use functionality like interceptors and Lisp advise. It also incorporates techniques from reflection, multiple inheritance, multi-methods and others. However, AOP has an explicit focus on crosscutting structure and the modularization of crosscutting concerns. The rule of thumb to avoid mechanism errors? Remember that AOP is more than any one mechanism–it’s an approach to modularizing crosscutting concerns that’s supported by a variety of mechanisms, including pointcuts, advice and introduction.”

(Kiczales, 2004)

It is also interesting to note some drawbacks about our approach:

- **Performance:** the runtime-based interceptions of method calls was heavy, adding a 10-factor more processing time at each method execution. However, since our system was not designed for high number of users or transactions per second, this was not a showstopper, at least not in that phase of the project.
- **Debugging:** the Visual Studio .NET 2002 debugger (used at the beginning of the project) had some trouble figuring out the “jumps” between the “normal” methods and the AOP code (which we called “hyperspace” in our jargon, since the call stack of the weaved code seemed to appear and disappear completely, almost magically, while debugging). The workaround was to set up breakpoints in strategic places around the code.
- **Microsoft support:** while .NET provides basic AOP capabilities, such as message interception and context-bound objects, these features were undocumented and explicitly non-supported; however, the code created back in 2002 still compiles perfectly well in later versions of the .NET Framework.

Conclusion

AOP helped us to have an extremely high productive environment, and in only 4 months we had a working system (4 people were work-

ing full-time in the project), providing an enourmous amount of functionality, and enhancing the current system.

Unfortunately, though, the project could not be finished completely, since the venture capital group that financed the company cut all financing unexpectedly and all development tasks stopped in 2003. Nevertheless, it was a highly rewarding experience, that gave me a practical insight into complex OOP, AOP and architectural design, as well as into project management techniques.

References

Fowler, Martin, “Inversion of Control Containers and the Dependency Injection pattern” [Internet] http://www.martinfowler.com/articles/injection.html (Last accessed July 2nd, 2006)

Hilsdale, Erik; Kiczales, Gregor et al., “Aspect-Oriented Programming with AspectJ©”, Xerox Corporation, http://www.ccs.neu.edu/research/demeter/course/w03/lectures/lecAspectJ-w03.ppt (Last accessed July 2nd, 2006)

Jacobson, Ivar; Ng, Pan-Wei, “Aspect-Oriented Software Development with Use Cases”, ISBN 0-321-26888-1, Addison-Wesley, 2005.

Kiczales, G., Lamping, J., Mendhekar, A., Maeda, C., Videira Lopes, C., Loingtier, J.-M., and Irwin, J, “Aspect Oriented Programming”, Springer-Verlag, 1997, [Internet] http://www.cs.ubc.ca/~gregor/papers/kiczales-ECOOP1997-AOP.pdf (Last accessed July 2nd, 2006)

Kiczales, Gregor, “Common Misconceptions”, Dr. Dobb’s Magazine, February 10th, 2004, [Internet] http://www.ddj.com/showArticle.jhtml?articleID=184415113 (Last accessed July 2nd, 2006)

Shukla, Dharma; Fell, Simon; and Sells, Chris, “Aspect-Oriented Programming Enables Better Code Encapsulation and Reuse”, MSDN Magazine, March 2002 [Internet] http://msdn.microsoft.com/msdnmag/issues/02/03/aop/ (Last accessed July 2nd, 2006)

Peer to Peer: An Overview

Adrian Kosmaczewski

2007-03-29

“P2P” or “peer-to-peer networking” is maybe one of the most controversial and interesting trends in the dawn of the new century; its complete decentralization challenges all definitions of private intellectual property and creates new technological, commercial and juridic challenges.

In this article I will enumerate some companies that provide P2P file-sharing services, after providing a short overview of the concept and implementation of P2P.

Technology

In a few words, the term peer-to-peer

> refers to the concept that in a network of equals (peers) using appropriate information and communication systems, two or more individuals are able to spontaneously collaborate without necessarily needing central coordination.

(Schoder & Fischbach, 2006).

In this broad category not only enter the well-known (and hated, see “The Register” reference below) file sharing networks, some of which will be enumerated later; but also applications old as the Internet itself, such as the Usenet or FidoNet, which can also be called P2P networks (Wikipedia, 2006).

[1]

As mentioned, file-sharing is not the only application of P2P networks; another interesting use of P2P technology is, for example, SubEthaEdit[2]. It is a collaborative text editor targeted to software developers, working in teams applying Agile methodologies such as XP (Extreme Programming)[3].

SubEthaEdit allows developers to work on the same document simultaneously, editing it and changing it as they see fit, and allowing to do peer programming when both developers are not located in the same physical place. It works only on Mac OS X 10.3.9 and above, since it depends on the Bonjour networking technology[4] created by Apple, but recently released as open-source[5].

Companies

Not being a user of P2P file-sharing networks myself, thanks to an article in P2P Blog[6] I've been able to quickly find a good number of companies[7] offering such services. From that list, I've selected the following 5 companies (I want to point out that I haven't personally tested any of these programs, and the summaries I provide are just extracted from the referenced websites):

[1] https://www.flickr.com/photos/rocketraccoon/227241974/
[2] http://www.codingmonkeys.de/subethaedit/
[3] http://www.extremeprogramming.org/
[4] http://www.apple.com/macosx/features/bonjour/
[5] http://developer.apple.com/opensource/internet/bonjour.html
[6] http://www.p2p-blog.com
[7] http://www.p2p-blog.com/item-187.html

- BitTorrent[8] is one of the most well-known systems for P2P file sharing. The home company is located in San Francisco, and it claims that its P2P software has been downloaded over 70 million times already. The software itself is available for Linux, Mac and Windows computers. Several third party clients are able to connect to BitTorrent networks, such as iSwipe[9] or Azureus[10]. An extensive list of BitTorrent clients is available here.[11]
- LimeWire LLC[12] is a New York-based company offering a client for sharing files through the Gnutella network. The client is available for Windows, Mac OS X, Mac OS 9, Linux, OS/2 and Solaris. There are no restrictions on the type of files that can be shared, but the website prompts the user to agree that the software will not be used for copyright infringement prior to download.
- Zapr[13] is company founded in 2002 in Singapore, providing a file-sharing software available for the Windows operating system, since it's based on the .NET Framework (version 2.0). Users are able to share any kind of file, without restrictions of any kind, for free.
- Red Swoosh[14] offers a particularly interesting approach to file-sharing; users that own a website and post files on it, but do not want to have to pay extra website bandwidth, can "swoosh" their links (that is, publish them on the Red Swoosh network), and people are asked to use the Red Swoosh client to download the files. The home company is located in San Francisco, and does not limit the type of files that can be shared. This approach, web-based, is typical of new "Web 2.0" companies (O'Reilly, 2005) that have appeared lately on the startup landscape.
- Pando[15] is aimed to people that have to share large files in different ways than using e-mail or instant messaging. The Pando client runs under Windows and the Mac OS X, and also works using plugins for Skype or Yahoo! Messenger, and the only current limit is the maximum size of the shared file (1 GB); apparently there are not restrictions on file types. The home company was created in 2005 in the city of New York, and it also features a typical "Web 2.0" style.

Conclusion

It has been really interesting to make this review, because I have never used file-sharing programs. Moreover, I had not given a good look to the concept of P2P, and as it turned out, it happens to have

[8] http://www.bittorrent.com/
[9] http://www.hillmanminx.net/iswipe/index.html
[10] http://azureus.sourceforge.net/
[11] http://www.slyck.com/bt.php?page=2
[12] http://www.limewire.com/
[13] http://www.zapr.net/
[14] http://www.redswoosh.net/
[15] http://www.pando.com/

different and interesting uses, many of them going beyond the much press-hyped copyright infringement issues.

References

Apple Developer Connection, “Bonjour”, [Internet] http://developer.apple.com/opensource/internet/bonjour.html (Accessed November 26th, 2006)

O’Reilly, Tim: “What is Web 2.0”, [Internet] http://www.oreillynet.com/pub/a/oreilly/tim/news/2005/09/30/what-is-web-20.html (Accessed November 26th, 2006)

P2P Blog, “Map of P2P Companies”, [Internet] http://www.p2p-blog.com/item-187.html (Accessed November 26th, 2006)

P2P United website, [Internet] http://wiki.morpheus.com/~p2punited/ (Accessed November 26th, 2006)

Schoder, D. and Fischbach, K.: “Core Concepts in Peer-to-Peer (P2P) Networking”. In: Subramanian, R.; Goodman, B.: “P2P Computing: The Evolution of a Disruptive Technology”, Idea Group Inc, Hershey, [Internet] http://www.idea-group.com/downloads/excerpts/Subramanian01.pdf (Accessed November 26th, 2006)

The Register, “Record industry uploads 8,000 lawsuits”, [Internet] http://www.theregister.co.uk/2006/10/17/ifpi_worldwide_filesharing_lawsuits/ (Accessed November 26th, 2006)

Wikipedia, “Peer-to-peer” [Internet] http://en.wikipedia.org/wiki/Peer-to-peer (Accessed November 26th, 2006)

Happy Birthday Scripting News!

Adrian Kosmaczewski

2007-04-02

Scripting.com is today 10 years old. It is the oldest running weblog on the web, an incredible source of information through the years. The beat of the web, from web 1.0, through the dot-com-boom, and until now in times of Web 2.0.

You can download the entire archive of Scripting.com here.

What was I doing 10 years ago? I had my first webpage online in August 1996 (unfortunately I haven't kept a copy of it); these postings are invitations I sent to some newsgroups asking someone to take a look at it. Everybody was using Netscape 3 and Internet Explorer was virtually nonexistent. Java was available in version 1.1, I had a laptop running Windows 95, and a PowerBook 150 running Mac OS 7.5. On April 1st, 1997 I moved to my first own appartment, in Geneva. I started a life 10 years ago. Then came Buenos Aires, then Lausanne, and here I am. In the meantime, I've gone through many technologies, I've travelled here and there, and I even got married. All in all, a lot has happened. For good.

Adrian Paenza

Adrian Kosmaczewski

2007-04-07

Me acuerdo de haber visto a Adrian Paenza en la television, cuando yo era pibe en Buenos Aires, alla por los '80; escucharlo en la radio o en la tele, hablando de futbol o de cualquier otro deporte, era parte de la cultura televisiva del domingo a la tarde, cuando pasaban los resultados deportivos y se acababa el fin de semana. Era como verlo a Macaya Marquez, a Fernando Niembro o a Pancho Ibañez.

Cual no fue mi sorpresa (expresion rara no?) al saber que Adrian Paenza, que comenzo su carrera deportiva con Muñoz en el '66[1], es Doctor[2] en ciencias matematicas de la Universidad de Buenos Aires (!) y no solo eso, sino que es profe y escritor.

Justamente como escritor, el Dr. Paenza cada tanto publica articulos muy buenos en Pagina/12, y ademas publico dos libros, que se pueden bajar gratuitamente de la web de la Facultad de Ciencias Exactas y Naturales: **"Matematica… Â¿Estas Ahi?"** Primera[3] y segunda[4] parte.

Update, 2009-01-30: La tercera[5] y cuarta[6] partes tambien estan a disposicion de quien quiera leerlas!

[1] http://contexto-educativo.com.ar/2001/6/paenza.htm
[2] http://es.wikipedia.org/wiki/Adri%C3%A1n_Paenza
[3] http://mate.dm.uba.ar/~cepaenza/libro/LIBRO_PAENZA.htm
[4] http://cms.dm.uba.ar/cep/libro-e2.html
[5] http://cms.dm.uba.ar/cep/libro-e314.html
[6] http://cms.dm.uba.ar/cep/libro-e100.html

QNX

Adrian Kosmaczewski

2007-04-11

I wrote a paper about QNX[1] that you can download here[2]. Happy reading!

[1] http://www.qnx.com/
[2] qnx.pdf

AJAX Presentation

Adrian Kosmaczewski

2007-04-26

Today I have done a tongue-in-cheek presentation of AJAX to my colleagues, including the historic background, the technology and the buzzwords. It also includes some code samples, compatible with Firefox, showing how to implement a synchronous and an asynchronous connection with the XmlHttpRequest component.

I did the original presentation with Keynote, and all I can say is: WOW! This application is light years away from everything else. It took me longer to figure out what to do than to do it. Powerpoint is such a pain in the … that this was a welcome switch. I just exported it to PDF for sharing it with you.

You can download everything right here! Feel free to tell me what do you think in the comments below.

Not Exactly What I Meant

Adrian Kosmaczewski

2007-05-02

This is a rant.

When I decided to leave behind all my years as a Microsoft developer and embrace open source, I wasn't exactly thinking about this. But hey, that's how it goes. At my job I'm working on a Java project; and… yes, exactly, you can imagine my face while writing this. And yes, this is my first Java project. Ever.

To summarize, this is just plain horrible; sensible hearts please don't continue.

[1]

To start locally this (fairly complex) application you have to click on IntelliJ IDEA's green arrow button, which is really straightforward after all. Then… you must wait for 4 minutes of compilation of changed files, then for the creation of JARs, the packaging of the WARs, their deployment, then there's Log4J starting, and also Tomcat starting

[1] https://www.flickr.com/photos/goldberg/265392460/

(where's Jerrymouse???), than wait to start some Catalina thing (it's funny, that's the name of a flower plant I had at Buenos Aires - and yes, I give names to my plants) and then, somehow, when you see no more activity on the messages window, you have to guess that you can open a browser and see your app in http://localhost/whatever. Thanks for telling me, I waited for 2 minutes staring on the message pane without understanding.

(Of course, I will skip the complete IntelliJ IDEA configuration; it took a colleague about half an hour to click around 10 different configuration screens in order to get my local copy working. If you think that's OK, well probably you value your time much less than I value mine, and you should stop reading.)

Needless to say, I said to myself; cool, now that everything's running, I'll add a JSP file and start playing with it. I created a folder, added a "helloworld.jsp" file to it, and pointed my browser to it.

404, not found.

OK, well, maybe I misplaced the file. Moved it to a place where a pre-existing html file is located. Retry in the browser. Nada. Rename to "helloworld.html". Neither. Verified the only line of Java code to see if that was the problem.

And then the coin dropped. I restarted Tomcat, and (5 minutes later) I could execute the file.

Naturally, I tried to modify the file; file/save and go to the browser. The same output as before. Hmmm... of course, if I restart Tomcat... my modifications appear.

WTF???

I cannot believe that in 2007, having used a couple of web application frameworks (to name a few: ASP, ASP.NET, PHP, Ruby on Rails, WebObjects) I happen to come accross the path of the stupidiest, clumsiest and most unnecessary complex way to run a web application, which by the way seems to be the standard way of doing stuff in this application stack. I am sorry, this is a rant and I told you. I'm outraged! Who the hell came up with this, and convinced developers around the world that it was worth it?

I can hear Java developers typing furiously in their keyboards while calling me names, because I am not able to configure Tomcat properly, after reading the gazillion pages of documentation available on the website. Well let me tell you: **I DO NOT CARE**. A technology that does not offer a coherent learning + approach path, just because of some sort of religious feeling of superiority or portability or security or I do not know which s**t, well, I know that I can press delete on that kind of technology fairly fast.

And yes, even using ASP.NET is easier, **much easier**. The OS detects that the file changed, recompiles it as soon as the first request arrives and serves the new content on the fly. I am not saying that ASP.NET is the way to go (far from that!) but there's at least one aspect that ASP.NET does better than this Java thing: **shorter development cycles** and **deployment**. I can see this now, very clearly. But even in the Java world, even playing with WebObjects was a much, much, much more pleasing experience, with shorter development cycles.

Is it that difficult to offer such a developer experience on the first place? Is it that complicated to learn that things should be easier? Is it that complicated to do things right from the beginning?

I really do not understand. I just wonder how long it will take for me to calm down.

UPDATE (May 3rd, 2007): I finally decided to let go the JSP way, and use PHP instead; for that, just use an Apache web server on port 8080, nicely configured with the PHP module, and using the proxy modules I can just redirect my requests to the Tomcat server running locally on port 8088; my httpd.conf file looks like this:

```
Listen 8080
...
LoadModule proxy_module modules/mod_proxy.so
LoadModule proxy_connect_module modules/mod_proxy_connect.so
LoadModule proxy_http_module modules/mod_proxy_http.so
...
DefaultType text/html
...
PHPIniDir "C:\\Program Files\\PHP\\"
LoadModule php5_module "C:\\Program Files\\PHP\\php5apache2_2.dll"
...
ProxyRequests On
ProxyVia On

<Proxy *>
Order deny,allow
Deny from all
Allow from localhost
</Proxy>

ProxyPass         /app http://localhost:8088/app
ProxyPassReverse  /app  http://localhost:8088/app
```

Et voilà! It worked from the beginning, and it only took 5 minutes to set up. Now I can use PHP files instead of JSP files, and I avoid restarting Tomcat every so often.

UPDATE (May 4th, 2007): Of course I forgot to add that the only means I've found so far to STOP this application is **to kill the java.exe process using Task Manager**. No comments.

Vive La France

Adrian Kosmaczewski

2007-05-06

La France est un pays immature. Tout aussi immature que l'Argentine et la Bolivie, d'après ce que je vois à la télé ce soir. La différence étant qu'en Argentine et en Bolivie on sait qu'on est immatures; on le sait et notre histoire est pleine de nos erreurs, terribles erreurs qui ont coûté la vie à des millions, des nôtres et des autres.

Et comme on sait tout ça, on ne se prend pas la tête avec des questions de "Liberté, La France est un pays immature. Tout aussi immature que l'Argentine et la Bolivie, d'après ce que je vois à la télé ce soir. La différence étant qu'en Argentine et en Bolivie on sait qu'on est immatures; on le sait et notre histoire est pleine de nos erreurs, terribles erreurs qui ont coûté la vie à des millions, des nôtres et des autres.

Et comme on sait tout ça, on ne se prend pas la tête avec des questions de "Liberté, Égalité, Fraternité" ni des propos sur la "grandeur de la France", ou d'autres conneries du genre.

Malgré tout, mine de rien, on arrive à croire que les choses peuvent être différentes, vraiment différentes, et que le changement est possible, lorsqu'Evo Morales arrive au pouvoir et donne son discours d'investiture en Quechua, sa langue maternelle. Il a été le premier président bolivien à le faire, dans un pays où la grande majorité des gens ne parlent pas l'espagnol.

Aujourd'hui, en France, un **fasciste**[1] **démagogue**[2] monte au pouvoir après avoir été ministre de la destruction pendant des années.

Si j'étais hypocrite, je dirais que "la démocratie a gagné" ou une autre connerie similaire. Ce n'est pas mon genre; je dis que la France est un pays immature, qui a une mémoire aussi courte que celle de n'importe quel autre pays. Un pays qui vote pour la violence tacite, la haine et la segrégation, au lieu de l'intégration, la paix et le pluralisme.

Dans le monde de l'informatique on se demande si on n'est pas de nouveau dans la bulle de 1999; eh bien, je me demande si aujourd'hui on n'est pas de retour en 1933.

[1] /blog/blogalite-ilegalite-fascistite/

[2] /blog/france-criminalizes-citizen-journalists-by-cellular-news/

Il y a deux ans, je demandais aux habitants de Londres d'ouvrir les yeux[3]. Je demande aux Français de se demander sérieusement qui est leur ennemi, et de ne pas laisser leurs libertés disparaître pour protéger ce concept flou qu'est devenu la **sécurité**[4].

[3] /blog/open-letter-to-the-people-in-london/
[4] http://www.odebi.org/new2/?p=281

Offshoring Quality

Adrian Kosmaczewski

2007-05-11

IT offshoring is a highly controversial topic nowadays; organizations from medium to large sizes choose to have their projects developed overseas, usually in countries where the cost of software development is lower than in Western Europe, Japan or North America.

Destinations

Several commonly chosen destinations for IT offshoring include (Source: OffshoreExperts.com):

- India
- Eastern Europe (Russia, Romania, Poland)
- Far east (China, the Philippines)
- Latin American countries (Mexico, Costa Rica, Brazil, Argentina)

However, other less-common offshoring destinations are Spain, Tunis, Kyrgyzstan, Zimbabwe or Marocco. In the Dilbert comic strip, Scott Adams has created the fictional country of “Elbonia” to satirize the situation created by offshoring in the IT landscape:

(Source: http://www.the-bordello.com/elbonia2.gif)

Advantages and Disadvantages

As I said at the beginning of the article, offshoring software development is a highly controversial topic, from several points of view, not only economically and socially speaking, but also from the point of view of the quality reached by companies using offshore (or near-shore) development teams. In short, what I’ve seen so far is that the

quality reached by offshoring software development is just plain horrible; however, the problem is seldom the overseas team itself, but rather the company headquarters employing the remote team.

This situation has to do with an inherent characteristic of software: it is not a manufactured product, but rather a design product. In other words, software is much more than the source code, but it is also a set of design decisions and multiple outputs, not only the source code itself, but also the documentation or the test and maintenance policies:

> This isn't the first time companies have tried to commoditize software development. In the eighties, Japanese companies unsuccessfully attempted to set up software factories to manufacture programs. They discovered that just throwing a lot of programmers together doesn't create innovative software.
>
> Design is a small part of clothing production, but a big part of software production. Unlike software, it makes sense to outsource the manufacture of clothing and toys. Most of the cost of clothing and toy manufacturing is in the assembly, not the design. Those products can still be designed close to corporate headquarters but assembled elsewhere to keep costs low.
>
> Programming is like design and nearly all of the costs of creating software come from writing the program, not the assembly. The assembly stage for software is really just copying the final program onto a disk and enclosing it with a manual in a box. (Bean)

Joel Spolsky stresses this fact even further:

> The best advice I can offer:
>
> **If it's a core business function – do it yourself, no matter what.**
>
> Pick your core business competencies and goals, and do those in house. If you're a software company, writing excellent code is how you're going to succeed. Go ahead and outsource the company cafeteria and the CD-ROM duplication. If you're a pharmaceutical company, write software for drug research, but don't write your own accounting package. If you're a web accounting service, write your own accounting package, but don't try to create your own magazine ads. If you have customers, never outsource customer service.

(Spolsky)

I have worked in some companies where the use of overseas offshoring is the rule; my impression is that the development cost reductions that these companies achieve are spent in communication, inspection and even turnover costs; until now, I have not yet seen a

single company working flawlessly with their offshore teams, no matter what the CEOs of these companies say.

The chosen destinations were Spain, India and Romania, and through this experience I could identify some factors that acknowledge the points of view expressed by Bean and Spolsky above:

- **Geographical location and time difference:** the quality of the end product is inversely proportional to the time zone difference between the headquarters and the remote team; the higher the difference, the lower the quality. This is simply due to the difficulty of having direct overlapping business hours between both parties, which makes communication more difficult. This also makes development cycles longer; the development-deployment-testing-feedback cycles are severely affected by low frequencies in the communication.
- **Cultural differences:** different holiday dates, different languages (including a variable level of English), all of these factors have a direct impact in the communication, which is a critical factor when dealing with outsourced teams.
- **Increased telecommunication costs:** the use of videoconferencing, telephone and other means brings new costs, even if these tend to decrease quite fast nowadays. The final quality of the end product is directly proportional to the smoothness and frequency of the communications between the headquarters and the remote team. There is also a need for distributed project management tools, including dashboards and other reporting functions, some of which are available as open source & free software, but usually the commercial packages that provide this functionality in an integrated fashion tend to be very expensive (like Microsoft's Visual Studio Team System or Borland's StarTeam suites).
- **Need for defined specifications:** the use of outsourced teams places a strong emphasis in the quality of the design and architecture documentations, which, more often than not, are simply not as accurate or complete as they should be. The final quality of the delivered product is directly proportional to the level of detail of the specification: the sloppier the documentation provided, the lower the quality of the received product, because the remote team, given the communication problems, tends to guess the missing details of the documentation in order to meet the deadlines.
- **Stronger verification and validation:** there is an increase in the costs of validation and verification, to make sure that the output received from overseas is adequate or correct. Part of the economies made in development must be reinvested in these processes, as a cost that is not always foreseen.
- **High turnover and bad working conditions:** in Argentina, where the IT offshoring sector is booming, the developers are working in extremely bad conditions, some of which would make the authors of "Peopleware" frown in dismay (Gremio de Infor-

maticos). In my country of origin, IT workers are usually working without contracts or social benefits, for salaries that are as low as those of other sectors (even for people with Master or PhD degrees), and they can be fired without explanation from their jobs. Even worse, the biggest companies that provide IT offshoring receive grants from the government, as an incentive for the creation of this kind of companies (jperezsistemas). This situation generates an extremely high level of turnover, which has a direct impact in the level of quality provided to the consumers of these outsourced services. I do not know if this situation happens in other countries, though.

Conclusion

In my opinion, the economies generated by offshoring, given a certain level of acceptable quality, are only visible under certain conditions:

- Large projects (usually measured in millions of dollars and hundreds of thousand lines of code);
- Small time zone and cultural differences between client and provider;
- Complete and rigorous design documents and specifications provided.

In all the other cases, I think that the situation is simply “Dilbertesque”.

Resources

Bean, M.; “The Pitfalls of Outsourcing Programmers - Why Some Software Companies Confuse the Box with the Chocolates”, [Internet] http://forio.com/resources/the-pitfalls-of-outsourcing-programmers (Accessed May 5th, 2007)

Dilbert cartoon (by Scott Adams) taken from http://www.the-bordello.com/elbonia2.gif (Accessed May 5th, 2007)

Gremio de Informaticos (Computer Workers’ Union of Argentina) website, [Internet] http://www.gremiodeinformatica.org.ar/ (Accessed May 5th, 2007)

jperezsistemas, “Como a EDS, Pyme tecnológicas piden que les subsidien el empleo”, [Internet] http://jperezsistemas.wordpress.com/ (Accessed May 5th, 2007)

OffshoreExperts.com, “Offshore Outsourcing Countries”, [Internet] http://www.offshorexperts.com/index.cfm/fa/buyer.countries (Accessed May 5th, 2007)

Spolsky, J.; “In Defense of Not-Invented-Here Syndrome”, [Internet] http://www.joelonsoftware.com/articles/fog0000000007.html (Accessed May 5th, 2007)

Why do I prefer OpenOffice, NeoOffice, or AbiWord?

Adrian Kosmaczewski

2007-05-17

I use OpenOffice[1] (for Windows and Linux), NeoOffice[2] (on the Mac) and AbiWord[3] (in the three OSs) more often than Microsoft Office, and I many people recently asked me why. My choice has to do with many reasons, that might or might not apply to you:

- They are **open-source**; not that I have compiled my own version or committed patches, but at least using them I somehow thank all the guys that spent hours getting this software right. And boy they did.
- They are **free**. You get an extraordinary value for nothing. Nada. Zero.
- **OpenOffice and NeoOffice work extraordinary well**. Version 2 of OpenOffice / NeoOffice is really stable and has a feature set similar to that of Microsoft Office 1997, which is far more than what I need anyway. So far I've had more crashes with Office 2003 and 2004 for Mac than with OpenOffice and NeoOffice, with a comparable usage. Many people who had used (and disliked) the previous versions should give it a try now; they might be surprised (as many friends of mine did recently!)
- It is **cross-platform**; I use Linux, Mac and Windows systems, sometimes simultaneously, and I need to read, write and share documents in the three platforms; OpenOffice, NeoOffice and AbiWord use the same document formats in the three platforms, and can open and read Microsoft Office documents as well.
- I tend to do documents with **diagrams**, and the drawing module of OpenOffice / NeoOffice is very handy and easy to use.
- **I do not use many Excel files with "macros"**; I know that this is a big problem for many people, since OpenOffice / NeoOffice does not do a good work of "translating" Excel to Calc macros; maybe it'll get fixed in the future, but it's not a problem for me.
- I prefer to use Keynote[4] for creating my presentations, which does not happen that often anyway. OpenOffice Impress has a

[1] http://www.openoffice.org/
[2] http://www.neooffice.org/
[3] http://www.abisource.com/
[4] http://www.apple.com/keynote/

similar workflow to PowerPoint, and I prefer a tool like Keynote, much more targeted to create stunning, fun, eye-catching presentations[5], than the usual bullet-point stuff that PowerPoint tends to have you do. However, Impress opens the occasional PowerPoint files that I receive, and in case of a problem, I prefer using a tool like SlideShare[6] to see and share them.

- **Integrated one-button PDF export**. This is very handy, and you do not need an external plugin for that. And of course, all the programs in the OpenOffice / NeoOffice suite can export documents in a lot of different formats, even RTF, LaTeX, SWF, SVG, JPG, PNG or HTML!

Personally, I do not see a reason use Microsoft Office anymore; actually in my job we use OpenOffice / NeoOffice as well, as the standard office suite, and I have uninstalled any version of Office Mac in my own computer at home. **I just don't need another Office suite; OpenOffice, NeoOffice and AbiWord are perfect for me in every sense.**

Of course, the same reasoning applies to Gimpshop[7], Inkscape[8] or VLC[9], for that matter!

[5] http://presentationzen.blogs.com/presentationzen/2005/10/the_lessig_meth.html
[6] http://www.slideshare.net/
[7] http://plasticbugs.com/?page_id=294
[8] http://www.inkscape.org/
[9] http://www.videolan.org/vlc/

15 Startup Commandments by Mark Fletcher

Adrian Kosmaczewski

2007-05-25

Read in startupping.com[1].

1. Your idea isn't new. Pick an idea; at least 50 other people have thought of it. Get over your stunning brilliance and realize that execution matters more.
2. Stealth startups suck. You're not working on the Manhattan Project, Einstein. Get something out as quickly as possible and promote the hell out of it.
3. If you don't have scaling problems, you're not growing fast enough.
4. If you're successful, people will try to take advantage of you. Hope that you're in that position, and hope that you're smart enough to not fall for it.
5. People will tell you they know more than you do. If that's really the case, you shouldn't be doing your startup.
6. Your competition will inflate their numbers. Take any startup traffic number and slash it in half. At least.
7. Perfection is the enemy of good enough. Leonardo could paint the Mona Lisa only once. You, Bob Ross, can push a bug release every 5 minutes because you were at least smart enough to do a web app.
8. The size of your startup is not a reflection of your manhood. More employees does not make you more of a man (or woman as the case may be).
9. You don't need business development people. If you're successful, companies will come to you. The deals will still be distractions and not worth doing, but at least you're not spending any effort trying to get them.
10. You have to be wrong in the head to start a company. But we have all the fun.
11. Starting a company will teach you what it's like to be a manic depressive. They, at least, can take medication.
12. Your startup isn't succeeding? You have two options: go home with your tail between your legs or do some-

[1] http://www.startupping.com/forums/showthread.php?t=347

thing about it. What's it going to be?

13. If you don't pay attention to your competition, they will turn out to be geniuses and will crush you. If you do pay attention to them, they will turn out to be idiots and you will have wasted your time. Which would you prefer?
14. Startups are not a democracy. Want a democracy? Go run for class president, Bueller.
15. You're doing a web app, right? This isn't the 1980s. Your crummy, half-assed web app will still be more successful than your competitor's most polished software application.

Poulet a La Portugaise: Menu Pour Nerds

Adrian Kosmaczewski

2007-05-27

Aussi connu comme “Pollo a la Portuguesa”. Une typique recette argentine (oui oui, je vous assure).

Ingredients (pour 8 personnes)

- 3 poulets entiers
- 2 grandes boites de tomates pelees et coupees
- 1 oignon, 1 gousse d’ail
- 1 grande boite de petits pois
- 1.5 Kg de pommes de terre
- Sel, poivre, huile

Preparation

Vous devez lancer 3 threads d’execution:

1. Dans une grande casserole, faire revenir l’oignon et la gousse d’ail dans l’huile: y ajouter les tomates et faire cuire doucement (doucement j’ai dit)
2. Dans une autre casserole, faire bouillir les pommes de terre
3. Entretemps, couper les poulets en 4 et enlever la peau (celle du poulet, donc)

Maintenant, vous faites des **.join()** entre les threads:

- Faire cuire la viande du poulet dans la sauce tomate
- Servir le tout avec les pommes de terre (on peut aussi servir avec riz, polenta, pizza, pasteques a l’ail, empanadas de guacamole, essence sans plomb ou batteries dechargees, mais c’est moins bon)

Bon app!

What Happened to Dr. Dobb's?

Adrian Kosmaczewski

2007-06-01

I have been a big fan of Dr. Dobb's[1] magazine for years. **Until last year.** Somehow during the month of June 2006, the magazine changed completely. I have read it since the late '90s, and I always found it to be at the bleeding edge of the technology; the articles were amazing, and the best of the best developers always wrote good descriptions of their discoveries and ideas. I much prefer reading in paper than on screen, and that's why I got a subscription to it.

However, right now I think it became a stripped-down version of MSDN Magazine[2]. The same full-page M$ ads, lots of articles about .NET and just a few about other technologies, and also a much, much lower article quality. They are uninteresting, even boring sometimes, and definitely not worth the original Dr. Dobb's that I knew.

I will be dropping my subscription to Dr. Dobb's as soon as it expires at the end of this year, and will take a subscription to IEEE Software[3] instead. Of course if you get to know another good software magazine I'd be delighted to hear about it!

[1] http://www.ddj.com/
[2] http://msdn.microsoft.com/msdnmag/
[3] http://computer.org/software

About Corporate Politics

Adrian Kosmaczewski

2007-06-03

Politics are part of our daily life. Nevertheless, the word has got a bad reputation in the IT world (and elsewhere, too), thanks to famous failures and managed disasters, but the truth is that to succeed, projects need politics - and project managers should know it well.

In this posting, I will describe the theoretical and empirical direct relationship between organizational verticality, worker alienation and political struggles, all leading to poor project performances.

About the word

DeMarco and Lister talk about the meaning of the word “Politics” in chapter 34 of “Peopleware”, named “The Making of Community”:

> The science of making communities, making them healthy and satisfying for all, is called politics.(…) Aristotle included Politics among the five interlinked Noble Sciences that together make up Philosophy. The five are: - Metaphysics: (…) - Logic: (…) - Ethics: (…) - Politics: how we may logically extend Ethics to the larger group, the science of creating and managing such groups consistent with ethical behavior and logical recognition of the metaphysical entities - humans and the community made up of humans. - Aesthetics: (…) Aristotelian Politics is the key practice of good management.

(DeMarco and Lister, page 223)

In our 21st century world, having a job means spending at least 8 or 9 hours of your time with lots of people, with whom you may or may not be happy to work with. To work as a team, particularly in knowledge-intensive jobs as in IT teams, the construction of a community is fundamental. This construction is deeply rooted in cultural backgrounds, values, ethic principles, and varies from one country to the other.

But we live in a globalized world, and for both the good and the bad, some inherent aspects of it are present everywhere; some of them are described by Philosophy. Some other are inherent to the capitalist system, and for these, I would like to point out the term that Marx

uses to describe the separation between the worker and his work; “alienation”:

> Marx identified four specific ways in which alienation pervades capitalist society. The product of labour: The worker is alienated from the object he produces because it is owned and disposed of by another, the capitalist.(…) The labour process: The second element of alienation Marx identified is a lack of control over the process of production. (…) Our fellow human beings: Thirdly, we are alienated from our fellow human beings. This alienation arises in part because of the antagonisms which inevitably arise from the class structure of society. (…) Our human nature: The fourth element is our alienation from what Marx called our species being. What makes us human is our ability to consciously shape the world around us. However, under capitalism our labour is coerced, forced labour. Work bears no relationship to our personal inclinations or our collective interests.

(Cox, 1998)

The creation of a community means making everyone in a team aware of the project scope, about the possibilities, about the vision and mission; everyone must buy it, and up to a certain level, feel responsible for it.

I think that the IT world is redefining the working rules of the capitalist society as we know it, and much of the problems about IT project management have to do with this particularly fundamental change in the worker/job paradigm.

Successful IT shops reduce alienation and encourage communities; they have, then, these particular characteristics, completely new in the management world:

- At the bottom, they tend to empower, rather than to limit, their teams; team members must feel that they own what they produce, in order to take responsibility for it
- At the top, management must learn to have a higher level of trust on their people; those in the field know things that management will never imagine. A good manager must “act like their most important job is to run around the room, moving the furniture out of the way, so people can concentrate on their work.” (Joel Spolsky, March 19th, 2000)
- Agility is part of the equation:

  > We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value: Individuals and interactions over processes and tools Working software over comprehensive documentation Customer collaboration over contract negotiation Responding to change over following a plan That is, while there is value in the items on the right, we value the

items on the left more.

(Agile Manifesto, 2001)

Daily Life

There are tons of anecdotes and examples about political conflicts in IT companies; my preferred is the series of articles that Joel Spolsky wrote, about the differences between Microsoft and Juno, in his blog "Joel on Software":

> Without going into too much detail, my manager decided he didn't like this. It became an issue of ego for him. First he yelled at the designer who was working on that page (without even telling me). Then he yelled at me. Then he reminded me every single day that I had to change it to the way he wanted it. Then he got the CEO of the company to review it, and made a big show out of getting the CEO of the company to criticize my new design. Even the CEO at Juno is perfectly happy to interfere in work done at the lowest level in the company, in fact, it's standard operating procedure.

(Spolsky, March 19th, 2000)

> For months later, we would have meetings where people would say things like "Charles [the CEO] doesn't like drop-down list boxes," because of something he had edited without any thought, and that was supposed to end the discussion. You couldn't argue against this fictional Charles because he wasn't there; he didn't participate in the design except for hit and run purposes. Ouch.

(Spolsky, March 23rd, 2000)

But I had also the opportunity of wrestling such "Command and Conquer" mentality in my own work as well.

Some time ago I was working for a big consulting firm in Geneva, and was told by my boss that the commercial teams had trouble figuring out the technical skills set of our consultants. The HR team did not have any database about us, at all, besides tons of paper CVs, which makes searching and filtering hard and inefficient, to say the least. Bid managers and other commercial staff members wanted a searchable database application in the intranet with that information, so they could match the incoming requests to the available skills in the company, and as such make bids and take better decisions. And of course, they wanted this application yesterday.

I proposed a quick solution using the Ruby on Rails framework (http://www.rubyonrails.org/) which I had been successfully using in personal projects; I was confident that such a tool could be done in less than 5 days, providing the needed functionality. My boss, on the other

hand, saw this project as the opportunity to show the hearquarters his commitment to the MDA (Model Driven Architecture, http://www.omg.org/mda/) methodology, which was pushed by the company's headquarters, as the standard way of doing things in our company. He made a clear statement about not wanting "toy" technologies in "his" department, and that he would not accept solutions outside of the boundaries of what the headquarters proposed.

It must be said that it was a tough year for our company, with several projects lost or failed. Time was running out and many members of my team had been fired. Needless to say, the human environment in the company was severely degraded. We all knew in the team that our boss was trying to get promoted, and that some of his wrong decisions were at the origin of some project failures in our team. To sum it up, the communication in our team, particularly between our boss and us, was broken after the layoffs.

Going back to the application, each of the solutions (Rails vs. MDA) had, of course, its long list of pros and cons, but my boss did not comment the CEO about other solutions than his own (I knew this by third people, afterwards). He got the approval and budget to create this application, and came back to me saying, basically, that I had to shut up and launch my UML editor.

This was enough for me to start the project by myself, at home, and in 3 days I had a working application, which I demoed to my peers in the office, and then proceed to set it up in one of our development servers. By the time I had the application up and running, the Java folks were just starting the first UML diagram needed to generate the application. Spontaneously, my teammates started entering their own curriculums in the application the same day (I was surprised) and came back to me with some ideas, bugs and modifications. I added the changes, commented the code out and uploaded everything to the internal version control system. The thing was ready.

Now for the interesting part: I showed the final version of the application to some commercial people, bypassing my boss, and that did it; they started using it almost immediately. They were impressed that we could deliver a solution that fast, and of course they went to my boss to congratulate him, which of course puzzled him. When he saw the thing working in the intranet, he went to the Java / UML team to congratulate them… They knew that I had been behind that one, and told my boss so.

He came back to my desk and asked me to remove the application from the server in that very moment. I refused, so he escalated this insubordination to the upper levels of the hierarchy, and when reaching the CEO, I suppose he was told to accept the thing, since I never heard about his opinion about this application again. The commercial teams started using it that same day.

A couple of months later I left the company. I heard that they are doing other applications with Rails right now, that the curriculums

application has been upgraded a couple of times, and that my ex-boss was fired, for other reasons.

Conclusion

The difference between success and failure in software companies has a lot to do with the “verticality” of the structures and the level of “alienation” of their workers; the traditional, military ways of management that lead to higher verticality and strong worker alienation do not work in the IT world, where knowledge, communication and collaboration are key to success.

I think that the rules of management are changing nowadays, and that this will ultimately turn into a major paradigm shift in our conception of the capitalist system. We need Politics, but as defined by Aristotle, not as Marx’s.

References

Beck, Kent, Thomas, Dave, Fowler, Martin et al., “Agile Manifesto”, 2001, [Internet] http://agilemanifesto.org/ (Accessed May 28th, 2006)

Cox, Judy, “An Introduction to Marx’s Theory of Alienation”, July 1998, International Socialism [Internet] http://pubs.socialistreviewindex.org.uk/isj79/cox.htm (Accessed May 28th, 2006)

DeMarco, Tom & Lister, Timothy, “Peopleware - Productive Projects and Teams, 2nd Edition”, 1999, Dorset House Publishing, ISBN 0-932633-43-9

Spolsky, Joel, “Command and Conquer and the Herd of Coconuts”, Joel on Software weblog, Thursday, March 23, 2000, [Internet] http://joelonsoftware.com/articles/fog0000000072.html (Accessed May 28th, 2006)

Spolsky, Joel, “Two Stories”, Joel on Software weblog, Sunday, March 19, 2000 [Internet] http://www.joelonsoftware.com/articles/TwoStories.html (Accessed May 28th, 2006)

Schedule Issues in Software Projects

Adrian Kosmaczewski

2007-06-05

Time is the ultimate dictator: “Time is also the one variable that has the least amount of flexibility. Time passes no matter what happens on a project.” (Schwalbe, chapter 6, page 202). As a result, any problem regarding the schedule of a project is likely to be the most difficult one to manage.

In this article I will enumerate some reasons that, in my opinion, make schedule problems a recurring cause of conflict in projects.

Factors

There are some factors that stress the influence of schedule issues in project conflicts; among these, I think that the most important are:

1. Work breakdown structure & empowerment;
2. Time perception;
3. Attitude, experience and authority of the project manager;

1) Work Breakdown Structure & Empowerment

As Kathy Schwalbe points out, “It is important for the project manager to empower project team members to take responsibility for their activities” (Schwalbe, chapter 6, page 233). Joel Spolsky goes simple and direct stating that “Only the programmer who is going to write the code can schedule it.” (Spolsky, 2000).

The key element here is, I think, to make all stakeholders aware of the proposed work breakdown structure, to analyze it, approve it, in order to make everyone agree on the sequence of tasks needed to achieve the project’s objectives. Particularly in the case of software projects, developers must be involved early in the project task decomposition.

2) Time Perception

It is a well-known fact that time perception is different for everyone, but this is particularly true for software developers. This may be hard to believe but it is real, and was even proven by scientific research.

DeMarco and Lister dedicated an entire chapter of their classic “Peopleware” book (chapter 10) to the discussion of time perception from the point of view of software development teams:

> During single-minded work time, people are ideally in a state that psychologists call flow. Flow is a condition of deep, nearly meditative involvement. In this state, there is a gentle sense of euphoria, and one is largely unaware of the passage of time: ‘I began to work. I looked up, and three hours had passed.’ There is no consciousness of effort; the work just seems to, well, flow.

(DeMarco and Lister, chapter 10, “Brain Time vs. Body Time”, page 63)

Inducing this “flow” has become the key element for the most successful software companies on Earth, including Microsoft, Apple and Sun, who take care of providing their employees with the quietest working environments:

> Strategy 5 Get Away From Interruptions
>
> (…) Look for ways to get out of this environment. Take a laptop to the company cafeteria, where there are lots of tables that are empty most of the day (and nobody can find you). Book a conference room for the whole day and write code there, and make it clear through the preponderance of checkins just how much more work you get done when you’re in a room by yourself.

(Spolsky, 2001)

This advice is obviously targeted for developers; for project managers, it can be useful to know that, “As well as trying not to be late, it’s also important to challenge the perception of lateness that other people may have of your project.” (Whitehead, chapter 11, page 82). But for this particular point, we must take a look at the personal qualities of the project manager.

3) Attitude, Experience and Authority of the Project Manager

Even after having taken extreme care of a project, schedule slippage may happen; this is a sad fact of life, but it happens. In this case, the communication and conflict management capabilities of the project manager will play a prominent role, since all the stakeholders will somehow resist change:

> Mantra: The fundamental response to change is not logical, but emotional.

(DeMarco and Lister, chapter 30, “Making Change Possible”, page 197).

The phrase above sums it all. The project manager needs to know how to tackle these situations, and for that, a good dose of interpersonal skills, coupled with strong conflict management capabilities, are fundamental.

The project manager has an enormous responsibility, and to be able to achieve the project's objectives, she / he must be able to align all stakeholders towards the same goal.

Conclusion

Several different factors generate most conflicts on projects: in my opinion, among them, project structure, psychological factors and political elements play a major role.

References

Joel Spolsky, "Painless Software Schedules", Wednesday, March 29, 2000 [Internet] http://www.joelonsoftware.com/articles/fog0000000245.html (Accessed April 29th, 2006)

Joel Spolsky, "Getting Things Done When You're Only a Grunt", Tuesday, December 25, 2001 [Internet] http://www.joelonsoftware.com/articles/fog0000000332.htm (Accessed April 29th, 2006)

Kathy Schwalbe, "Information Technology Project Management, Fourth Edition", Thomson Course Technology, 2006, ISBN 0-619-21526-7

Richard Whitehead, "Leading a Software Development Team - A Developer's Guide to Successfully Leading People & Projects", Addison-Wesley, 2001, ISBN 0-201-67526-9

Tom DeMarco & Timothy Lister, "Peopleware - Productive Projects and Teams, 2nd Edition", 1999, Dorset House Publishing, ISBN 0-932633-43-9

Application Frameworks

Adrian Kosmaczewski

2007-06-08

One of the most pragmatically influential changes in software development, since the late eighties, was the introduction of several object-oriented frameworks, in different programming languages. This article will provide a short overview of their impact in the software engineering field.

Application Frameworks - Definition

According to Wikipedia,

> In computer programming, an application framework is a term usually used to refer to a set of libraries or classes that are used to implement the standard structure of an application for a specific operating system. By bundling a large amount of reusable code into a framework, much time is saved for the developer, since he/she is saved the task of rewriting large amounts of standard code for each new application that is developed.

(Wikipedia, 2006)

Moreover, Booch describes application frameworks as

> At the high end of the food chain, we have frameworks. A framework is a collection of classes that provide a set of services for a particular domain; a framework thus exports a number of individual classes and mechanisms which clients can use or adapt. As we will discuss in Chapter 9, frameworks represent reuse in the large.

(Booch, 1994, page 166)

Most importantly,

> The primary benefits of OO application frameworks stem from the modularity, reusability, extensibility, and inversion of control they provide to developers

(Fayad & Schmidt, 1997)

Characteristics and Advantages

Frameworks usually consist of a large number (thousands) of classes, organized into packages, libraries or namespaces, which application developers can use in the same way as a child uses Lego bricks while playing. This way, application frameworks provide a high level of reuse thanks to:

- **Ad-hoc standardization:** some frameworks, such as Java or .NET became de facto standards thanks to the spread of their usage
- **Speed of development:** frameworks provide easy solutions and APIs (Application Programming Interfaces) to common problems, such as memory management, string manipulation, networking, I/O, etc, which are useful to a large range of applications
- **Lower costs and easier maintenance:** third-party frameworks are created and maintained by specialized companies, and updates are distributed usually without charge to customers, which means less maintenance costs for the industry as a whole.

Historical Evolution

Historically speaking,

> The first object-oriented frameworks to be considered as such are the MVC (Model View Controller) for Smalltalk and the MacApp for Apple applications. Other important frameworks from this initial phase were ET++[Weinand et al., 1989] and Interviews. It is interesting to note that most of the seminal frameworks were related to designing user interfaces. In the framework history, it is also important to cite the name of the Taligent company, a joint venture of Apple Computer, IBM and Hewlett-Packard. They developed a set of tools for rapid application development under the name of “Common Point’’ that consist on more than a hundred OO frameworks.

(Amatriain, 2004)

Another important step in the historical evolution of frameworks was the NeXTSTEP platform, which strongly influenced Java:

> When I left Sun to go to NeXT, I thought Objective-C was the coolest thing since sliced bread, and I hated C++. So, naturally when I stayed to start the (eventually) Java project, Obj-C had a big influence. James Gosling, being much older than I was, he had lots of experience with SmallTalk and Simula68, which we also borrowed from liberally.

(Naughton)

The similarities between NeXTSTEP (later known as Cocoa when Apple

and NeXT merged) and Java have made possible for Apple engineers to create a “bridge” that allows the Java language to access native Cocoa objects.

Known Frameworks

Some current examples of generic object-oriented frameworks include:

- Microsoft .NET (C#, VB.NET, C++, many others): http://msdn.microsoft.com/netframework/
- Cocoa (Objective-C, Java, Ruby, others): http://developer.apple.com/cocoa/
- Ruby on Rails (Ruby): http://www.rubyonrails.org/
- Java SDK and runtime: http://java.sun.com/
- Apache Struts (Java): http://struts.apache.org/
- Tapestry (Java): http://tapestry.apache.org/
- Webobjects (Java): http://www.apple.com/webobjects/
- Spring (Java and .NET): http://www.springframework.org/ and http://www.springframework.net/
- Prototype (JavaScript): http://prototype.conio.net/
- Dojo (JavaScript): http://dojotoolkit.org/
- Zend (PHP): http://framework.zend.com/
- CakePHP: http://www.cakephp.org/
- Seagull (PHP): http://seagullproject.org/
- Prado (PHP): http://www.xisc.com/
- Zoop (PHP): http://zoopframework.com/
- Django (Python): http://www.djangoproject.com/
- Catalyst (Perl): http://www.catalystframework.org/
- wxWidgets (C++, Ruby and others): http://www.wxwidgets.org/
- Qt (C++): http://www.trolltech.com/products/qt
- Seaside (Smalltalk): http://www.seaside.st/

The “Application Framework Project” of OpenOffice.org (http://framework.openoffice.org/) shows a good example of an application framework designed to serve a specific set of functionality, to a limited set of well-known clients.

Conclusion

Application frameworks provide a complete set of functionality that can be used “off the box” to solve software problems, achieving an extremely high level of reuse, and reaching standardization at the level of a whole industry.

References

Amatriain, Xavier; University of California Santa Barbara, “An Object-Oriented Metamodel for Digital Signal Processing”, October 2004 [In-

ternet] http://www.create.ucsb.edu/~xavier/Thesis/ (Accessed July 22nd, 2006)

Booch, Grady, “Object Oriented Analysis and Design With Applications, Second Edition”, Addison Wesley, 1994, ISBN 0-8053-5340-2

Fayad, Mohamed & Schmidt, Douglas C., “Object-Oriented Application Frameworks”, 1997 [Internet] http://www.cs.wustl.edu/~schmidt/CACM-frameworks.html (Accessed July 22nd, 2006)

Naughton, Patrick, “Java Was Strongly Influenced by Objective-C” [Internet] http://www.cs.umd.edu/~seanl/stuff/java-objc.html (Accessed July 22nd, 2006)

Wikipedia, “Application Framework”, [Internet] http://en.wikipedia.org/wiki/Application_framework (Accessed July 22nd, 2006)

Users as Testers

Adrian Kosmaczewski

2007-06-24

Both users and black-box testers approach software with apparently similar attitudes; however, they have fundamentally different goals. With this article I will try to expose some of my ideas about this dichotomy.

Users and Testers

When it comes to black-box software testing, there are two main differences between users and testers, in my opinion; the first one, maybe the most obvious, is that the user usually paid for the software, while the tester is paid to use (and break) it. This is very important, since all efforts should be done to make that the final user gets the best possible experience, and the smallest possible number of bugs. After all, the user is paying for the software, and she needs it to fulfill some business problem or some particular need, be it run a company, play a game or write a mail.

The tester uses the software with the sole objective of **finding problems**. The user tries to **avoid them** at all price.

The second main difference, is that if your users become (by any chance) de facto testers of your software, it means that you have shipped buggy code, or even worse, you might consider that having your users test your product is the right thing to do. Both of which are bad ideas anyway:

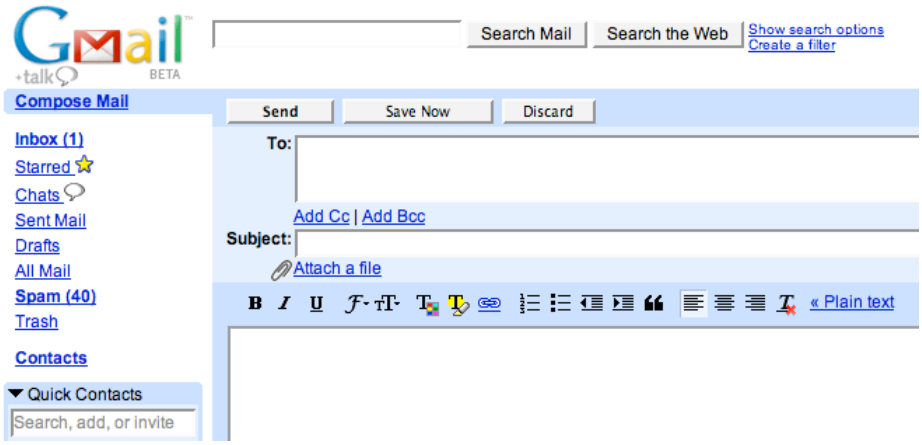
> The worst thing about this form of testing is the remarkably bad impression you will make of your company. When Userland released the first Windows version of their flagship Frontier product, I downloaded it and started working through the tutorial. Unfortunately, Frontier crashed several times. I was literally following the instructions exactly as they were printed in the tutorial, and I just could not get more than 2 minutes into the program. I felt like nobody at Userland had even done the minimum amount of testing, making sure that the tutorial works. The low perceived quality of the product turned me off of Frontier for an awfully long time.

(Spolsky, 2000)

The web brought a whole new medium for software deployment; through a web browsers, users were suddenly able to access new versions of applications almost instantaneously, and this brought the blurry line between users and testers to another level.

The “Eternal Beta” Situation

Since the late 90s, several companies have relied on users’ input to find flaws in their systems. Even nowadays, several popular applications used by millions of people are presented as “beta”; for example, Google’s webmail, Gmail:



(Source: Gmail.com, as of May 17th, 2007)

Gmail’s own “Terms of Use” specify clearly that

> In addition, you understand and agree that the Service is provided on an AS IS and AS AVAILABLE basis. Google disclaims all responsibility and liability for the availability, timeliness, security or reliability of the Service. Google also reserves the right to modify, suspend or discontinue the Service with or without notice at any time and without any liability to you.

(Gmail)

Through the above statement, Google is able to modify the appearance, functionality and usefulness of Gmail, in any way, at its sole discretion; in this environment, end users appear as ad hoc testers, sometimes making the big headlines:

> Seems like every couple of months reports surface about outages with Gmail (and Yahoo Mail too). Lately reports about Gmail accounts being disabled and people being unable to access their mail for hours, if not days, have been heating up. Google says it has received feedback from some Gmail users who were having trouble accessing their accounts but that the incidents were isolated and

> have been fixed. (…) Problems like these, even sporadic and eventually fixed, can damage the reputation of any Internet service provider, but particularly one that's increasingly marketing itself as the go-to company for hosted applications aimed at consumers and businesses.

(Mills, 2006)

This phenomenon has taken the web development industry as a storm:

> Following Google's lead, many companies stick "beta" on their logos and leave it there for months or years. Gone are the betas that get released to a limited set of known but external testers, with formal product management follow-up interviews. The concept of "beta" as a time period or stage of development has fallen away, and been replaced with beta as a way of setting expectations, or excusing faults, about the current state of the application.

(Hedlund, 2006)

And sometimes, this attitude is absolutely and completely explicit:

> We don't have the resources to do full usability tests, so you are our worldwide usability assessment.

(Jotlet Blog, 2006)

Conclusion

Black-box testers are an intermediate kind of user: they click on the user interface, launch processes and input data, and actually they try to get things done with the software like any other user would do; but their ultimate objective is not the same as that of the final user: they are discovering flaws before any user do, which will prove to enhance the perceived quality of the final product, drive up sales, increase the trust in the organization and lower the chances of project failure. And even better, all of these factors will have long-term effects, both for the producers and the clients of the final product.

References

Gmail, "Gmail Terms of Use", [Internet] http://www.google.com/mail/help/terms_of_use.html (Accessed May 17th, 2007)

Hedlund, M.; "Web Development 2.0", February 2006, [Internet] http://radar.oreilly.com/archives/2006/02/web_development_20.html (Accessed May 17th, 2007)

Jotlet Blog, "So what do you think?", September 2006, [Internet] http://blog.jotlet.net/?p=11 (Accessed May 17th, 2007)

Mills, E.; “Gmail problems... again?”, C|net News.com, March 23rd, 2006, [Internet] http://news.com.com/8301-10784_3-6053454-7.html (Accessed May 17th, 2007)

Spolsky, J.; “Top Five (Wrong) Reasons You Don’t Have Testers”, April 2000, [Internet] http://www.joelonsoftware.com/articles/fog0000000067.html (Accessed May 17th, 2007)

Ext

Adrian Kosmaczewski

2007-07-11

Should you ever have to work on a web application again, just don't think about it twice; Ext[1] is an amazing piece of free software, light-years away from anything you've seen before.

Ext is a JavaScript framework, unlike any other in the market. It consists of a great mix of high level API documentation, coherent coding style, and excellent support, going beyond the mere lines of code. It is the first time in 10 years of web development that I really find a web platform worth that name.

Ext is more than a framework: is the DOM as it should have been in the first place.

[1] http://extjs.com/

Code Coverage Using gcov

Adrian Kosmaczewski

2007-07-23

I've just uploaded a code coverage test project, using the gcov GNU tool. The idea was to create a small application (simulating an ATM), and injecting into the CppUnit unit tests executable code coverage information, using the gcov utility. And the results just speak by themselves:

```
       32:  301:    const bool Date::isLeapYear() const
        -:  302:    {
       32:  303:        if ( _year % 4 != 0 )
        -:  304:        {
       19:  305:            return false;
        -:  306:        }
        -:  307:        else
        -:  308:        {
       13:  309:            if ( _year % 100 != 0 )
        -:  310:            {
    #####:  311:                return true;
        -:  312:            }
        -:  313:            else
        -:  314:            {
       13:  315:                if ( _year % 400 != 0 )
        -:  316:                {
        2:  317:                    return false;
        -:  318:                }
        -:  319:                else
        -:  320:                {
       11:  321:                    return true;
        -:  322:                }
        -:  323:            }
        -:  324:        }
        -:  325:    }
```

The above code sample from the date.cpp.gcov file (in the project zip file) show that the unit tests run called 32 times the Date::isLeapYear() method, of which 19 were not leap years, and the rest were. The interesting bit is in line 311, **which was never called**, as shown with the "**#####**" sign! This is extremely nice, since it shows that my tests are not 100% comprehensive, and some cases are not tested.

On the downside, I must say that the gcov tool is not really easy to use (it took me a while to figure out how to do things) but grosso modo it works in the following way:

1. You must compile and link the unit test application (for example in “Debug” mode, or maybe other special ad hoc configuration) using the `-fprofile-arcs -ftest-coverage` GCC flags (in Xcode you can check the “Generate Test Coverage Files” and “Instrument Program Flow” option checkboxes) and the `-lgcov` linker flag. I also set Xcode to “ZeroLink”, told GCC to do optimization level “None [-O0]”, unchecked the option to generate position-dependent code, and disabled prebinding;
2. You run the application: this will generate a folder with a bunch of files with “.gcda” and “.gcno” extensions, together where the `.o` files are output during the build (when using Xcode on a PPC Mac, these files are created in build/atm.build/Debug/tests.build/Objects-normal/ppc);
3. Open a command line window and run the commands `gcov FILENAME.gcno` and `gcov FILENAME.gcda` and you will get, in the same folder, a file called `FILENAME.cpp.gcov` with the information shown in the snippet above. By the way, you get files for all the dependencies of that source code as well.

I hope this helps! Having this information can really help ensuring that test cases are comprehensive and complete. As always, if someone has a tip or a comment about gcov and wants to share it, I would be glad to read about it in the comments below.

Viacard

Adrian Kosmaczewski

2007-07-26

Desde hace unos meses que vengo siguiendo un blog argentino llamado Viajé como el orto[1], por sugerencia de hernún[2]. Básicamente, en Buenos Aires se viaja cada vez peor, y esto está llegando a niveles alarmantes, muy lejos de ser anecdótico.

Eso me hizo acordar de un péiper que escribí cuando estaba estudiando la materia "Sistemas Administrativos" en la UBA, donde describo un sistema de pago unificado para los transportes del Gran Buenos Aires, que empieza con la siguiente afirmación:

> La Ciudad de Buenos Aires está atravesando la mayor crisis imaginable de transporte urbano de su historia. Y la tendencia actual no muestra signos de mejoría en el corto plazo.

Esto lo escribí en el 2000.

Bueno, hurgando en unos viejos DVD de archivo en casa encontre el péiper en cuestión, lo pasé a PDF y lo puse en la seccion "proyectos"[3], para que la gente se lo baje y lo vea y si le sirve, mejor.

[1] http://viajecomoelorto.blogspot.com/
[2] http://hernun.com.ar/blogs/enta/
[3] viacard.zip

POSIX Device Files

Adrian Kosmaczewski

2007-07-27

Modern operating systems provide a clear separation of the kernel processes from those running in user space, which prompts the question of how to access I/O devices from user processes, without breaking the above mentioned architectural separation, which guarantees stability, security and performance.

Several approaches are available, depending on the level of abstraction used and the context of the problem. One of the solution is using POSIX Device Files, which indeed provide the same system-call interface for both files and devices. This article will describe the POSIX standard, the POSIX device files, and give a short enumeration of advantages and disadvantages of them.

The POSIX Standard

POSIX stands for "Portable Operating System Interface for Unix", and it is the "collective name of a family of related standards specified by the IEEE to define the application programming interface (API) for software compatible with variants of the Unix operating system." (Wikipedia). Nearly all of today's most important operating systems, in either closed or open source form, are POSIX-compliant to a certain degree: all versions of Microsoft Windows, Apple Mac OS X, OpenVMS and Solaris, for example, are fully POSIX-compliant, while Linux, FreeBSD and Nucleos RTOS are partially compliant (Wikipedia). POSIX Conformance Test Suites (for example the one provided by NIST[1]) are used to determine the level of POSIX-compliance of an operating system.

The importance of the POSIX standard is hard to assess; however, it is widely recognized that it has had an enormous impact in the computing world as we know it today:

> The overall Unix market (Figure 1) boasts billions of dollars in revenues, and is projected to continue growing into the future. Major areas for this growth feature high-end systems (data warehousing, servers, and supercomputing) rather than desktop applications. Brian Unter evaluated the

[1] http://www.itl.nist.gov/div897/ctg/posix_form.htm

> impact of Posix standards in this area as responsible for up to 30% of Hewlett-Packard's Unix business.

(Isaak & Johnson, 1998)

The POSIX standard provides the whole industry with a single, enormous and strong foundation, allowing interoperability, benchmarking and standardization, which in turn helps reducing costs, making the whole industry more efficient.

[2]

POSIX Device Files

One of the key aspects of the POSIX standard is the definition of a common gateway to access I/O devices from the user space, avoiding a direct and probably dangerous communication with the kernel, and known as POSIX Device Files: "These interfaces enable communication with serial, storage, and network devices through device files. In any UNIX-based system such as BSD, a device file is a special file located in /dev that represents a block or character device such as a terminal, disk drive, or printer. If you know the name of a device file (for example, disk0s2 or mt0) your application can use POSIX functions such as open, read, write, and close to access and control the associated device." (Apple)

What does all of this mean? In short, this means that

[2] https://www.flickr.com/photos/uwehermann/85855177/

> Under UNIX, every piece of hardware is a file. To demonstrate this, try view the file `/dev/hda`
>
> ```
> less -f /dev/hda
> ```
>
> `/dev/hda` is not really a file at all. When you read from it, you are actually reading directly from the first physical hard disk of your machine. /dev/hda is known as a device file, and all of them are stored under the /dev directory.

(Sheer)

Advantages

The main advantage is homogeneity. Using POSIX device files, developers can manipulate, read and write data from external I/O devices without having to directly access the operating system kernel, using a unified programming model, with a standardized set of semantics, which leads to systems designed with elegance and correctness.

Another advantage is tightly related to the concept of “streams”, as understood by many higher-level programming languages. Streams allow developers to access devices in an uniform way; C++, Java, the .NET languages, and even dynamic languages such as Ruby or Lisp allow to treat sequences of bytes using this uniform concept:

> A stream is an object that can be used with an input or output function to identify an appropriate source or sink of characters or bytes for that operation

(Lisp.org)

Java and .NET have big inheritance trees below the abstract Stream class, handling memory-based, file, network or I/O device-based streams, all of them sharing a similar structure and behavior, this means mainly the capability of being read and / or written in a sequential fashion.

This paradigm, coupled with the use of POSIX device files, allow software architects to create flexible designs, where the streams are treated polymorphically, can be exchanged at runtime as needed, and have lesser dependencies among them.

Disadvantages

The main disadvantage might be the maintainability of the code that uses these abstractions, since that these notation and semantics might not be immediately obvious to a newly graduated programmer.

The most common programming experience, in higher-level abstractions, usually depends on higher abstractions to access devices, for example using the System.IO.SerialPort classes in the .NET programming model:

> To reduce the programming effort that is required when working with serial ports, the .NET Compact Framework 2.0 includes the SerialPort class. The SerialPort class provides a simplified abstraction over serial communications ports that provides a number of features that simplify monitoring and configuring serial ports. The serial port also simplifies sending and receiving data with serial ports–including the automatic encoding and decoding of data sent to and received from the port

(Microsoft)

Mac OS X also provides higher-level abstractions for managing devices:

> A device interface is a plug-in interface between the kernel and a process in user space. The interface conforms to the plug-in architecture defined by Core Foundation Plug-in Services (CFPlugIn), which, in turn, is compatible with the basics of Microsoft's Component Object Model (COM). In the CFPlugIn model, the kernel acts as the plug-in host with its own set of well-defined I/O Kit interfaces, and the I/O Kit framework provides a set of plug-ins (device interfaces) for applications to use

(Apple)

The abstraction-level problem also arises when porting code from an operating system to another, since a common abstraction to access a CD-ROM drive in Windows (usually "D:") is different from the way that Linux does (usually"/cdrom") or the way that Mac OS X uses (which is using the friendly name of the CD-ROM under the "/Volumes" device). The use of POSIX device files might help avoiding these problems.

Conclusion

Both the advantages and disadvantages of using the same system-call interface are relative to the problem being solved, the available budget and the skills of the engineering team. There are huge advantages to stick to the POSIX standard, but there is a cost in maintainability and readability as well.

References

Apple Developer Connection, "I/O Kit Fundamentals, Architectural Overview: Controlling Devices From Outside the Kernel", [Internet] http://developer.apple.com/documentation/DeviceDrivers/Conceptual/IOKitFundamentals/ArchitectOverview/chapter_3_section_7.html (Accessed February 11th, 2007)

Isaak, J.; Johnson, L.; "Posix/Unix standards: foundation for 21st century growth", Micro, IEEE, Volume 18, Issue 4, July-Aug. 1998

Page(s):88, 87; Digital Object Identifier 10.1109/40.710874

Lisp.org, “System Class STREAM”, [Internet] http://www.lisp.org/HyperSpec/Body/syscla_stream.html (Accessed February 11th, 2007)

Microsoft, “What’s New in the .NET Compact Framework 2.0”, [Internet] http://msdn2.microsoft.com/en-us/library/aa446574.aspx (Accessed February 11th, 2007)

NIST, “PCTS:151-2, POSIX Test Suite”, [Internet] http://www.itl.nist.gov/div897/ctg/posix_form.htm (Accessed February 11th, 2007)

Sheer, P.; “UNIX devices”, [Internet] http://www.cacs.louisiana.edu/~mgr/404/burks/linux/rute/node18.htm (Accessed February 11th, 2007)

Wikipedia, “POSIX”, [Internet] http://en.wikipedia.org/wiki/POSIX (Accessed February 11th, 2007)

Web Development is Software Development

Adrian Kosmaczewski

2007-08-02

I have been developing web applications since 1996, and I still do a fairly large amount of web development nowadays. During these years I have seen some common misconceptions and myths about web development, that ultimately have a direct impact in the usability of the system. I will outline these in this article, providing at the end my opinion on how to achieve a proper QA process.

[1]

Website Development IS Software Development

A website is a piece of software, whether the team acknowledges the fact or not; as such, all the best practices for software development apply to web design and development as well, particularly in those projects that aim to build interactive “web applications” instead of simple “brochure websites” based on some CMS, usually displaying rather static information.

Acknowledging this fact is something that many teams do not do (or want to do); in my opinion, just because JavaScript lacks a compiler it does not mean that it is not a programming language, and sooner or later, nearly all websites use JavaScript - arguably the “world’s most misunderstood programming language” (Crockford). Thus, the following tasks apply even for the smallest website:

- Have a specification;
- Have a schedule;

[1] https://www.flickr.com/photos/tyger_lyllie/593557424/

- Have an architecture (albeit small) that separates the UI from the rest of the application; this allows the designers to change the look & feel without disturbing those working in the functionality;
- Make a prototype; preferably many of them; keep them for future reference;
- Have (human) testers;
- Automate your tests (and builds, if you use a compiled technology like Java);
- Use a bug database;
- Use version control;
- Be agile: get your client involved and have small development-testing-deployment cycles;
- Know your audience;
- Scope your target.

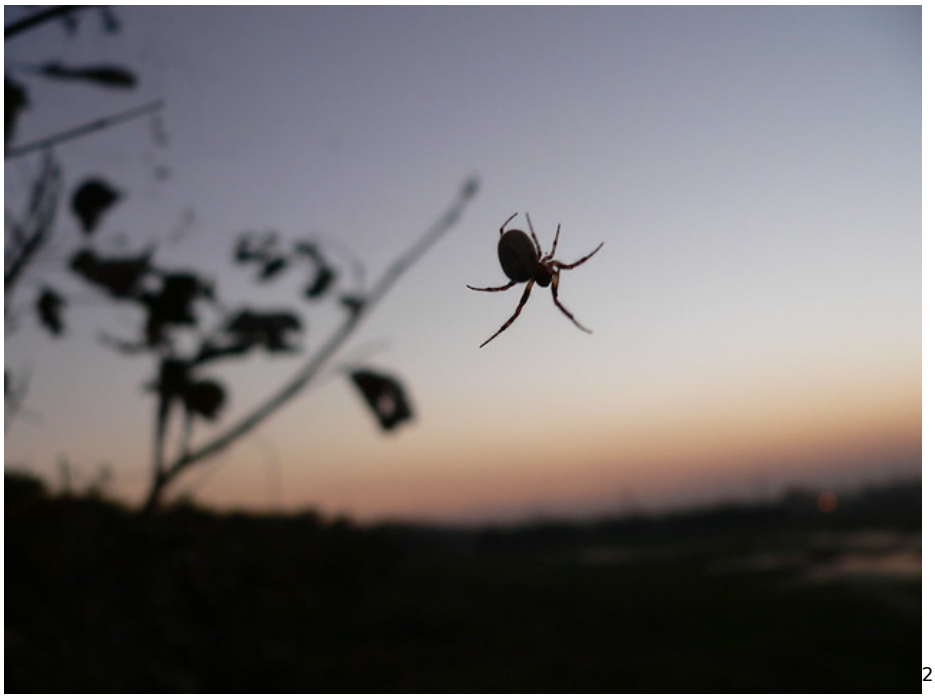
I will provide a small comment on the last two items in the following sections.

Know Your Audience

A very important point to know is the target audience of the website: is it a public or intranet system? What is the average age of the users? What are the legal and accessibility requirements? And last but not least, what languages should we support? These informations must come from a simple analysis of the target users, and the answers to these questions must be stated in the design document of the website project.

Knowing the target languages is fundamental, since it means choosing a target encoding for the system output; Unicode UTF-8 is a canonical choice, but for English-only websites ISO-8859-1 is more than enough. On the other side, some languages like Arabic are displayed in a “right-to-left” fashion, and this means that your application must look good and work properly in this mode too.

Finally, some clients (particularly governments and federal agencies) require providers to comply with accessibility rules, for supporting users with disabilities. The use of text-to-speech browsers, large font sizes and other media must then be taken into account during analysis, design and development of the application.

 [2]

Scope Your Target

When developing web applications, one common mistake that I've seen many development teams do is to forget to set up a list of minimum target browser requirements for the application. In my experience, relying on HTML, CSS or other standards is fundamental but sadly not enough: setting a scope means creating just a short list of browsers, including screen size, browser and operating system versions, scoping the sandbox where the QA operations will be applied. This list must be part of the specification of the system being created, and must be visible and known by everyone. A sample list could be the following:

- Internet Explorer 5.5+ for Windows XP Service Pack 2
- Firefox 1.5 and all Gecko 1.8-based browsers (many different OS)
- Konqueror 3 for Linux
- Safari 2 for Mac OS 10.3
- Opera 9 (any OS)

Using the name of the layout engine also helps (like the "Gecko" reference above) since several browsers use the same layout engine, providing the same characteristics to otherwise different browsers: for example, Gecko 1.8 is used in Mozilla, Camino (for Mac OS X), Galeon, Firefox 1.5 and 2.0, and other browsers (Wolter, 2007)

It is also important to scope the minimum supported screen size:

[2] https://www.flickr.com/photos/arulprasad/90211438/

- Minimum screen size: 1024 x 768 pixels.

Of course, if the website must support other types of devices (game consoles like the Wii or the Xbox, cell phones or PDAs), this must be specified as well, with detailed version information, and details about the supported screen sizes.

Last but not least, it is fundamental to define the "DOCTYPE" to be used in the website; this setting defines the set of valid HTML tags to be used in the page:

> Per HTML and XHTML standards, a DOCTYPE (short for "document type declaration") informs the validator which version of (X)HTML you're using, and must appear at the very top of every web page. DOCTYPEs are a key component of compliant web pages: your markup and CSS won't validate without them.

(Zeldman, 2002)

Another good practice is to add an explicit list of definitely non-supported browsers or operating systems:

- Netscape 4.x (pre-Gecko rendering engine)
- Opera (versions prior to 9)
- Internet Explorer for the Macintosh
- Mac OS versions prior to 10.3
- Linux kernels prior to 2.4

Perhaps surprisingly, these simple actions are often forgotten, which leads to confusion and lack of coherence in the QA activities of the team. Managers avoid having developers refusing to fix some incompatibility because of the lack of standards support in some browser (which always happens). On the other side, the whole team is shielded against some change in the scope; for example, the marketing team might come up with a requirement to support Opera 8, and this can have a negative impact on the schedule if it was not specified upfront, since all the CSS and JavaScript code might have to be tweaked to support the new browser.

Different Dimensions

As shown in the above paragraphs, websites are complex beasts, that can have several (often conflicting) dimensions:

- Different languages;
- "Right-to-Left" against "Left-to-Right" layouts;
- Different target browsers;
- Different target operating systems;
- Users with potential disabilities;
- Finally, the website functionality itself!

How to manage this complexity? Unfortunately, given the high fragmentation of the web market, there is not a simple answer to this

problem; I think that (at least currently) the best approach is a mix of automated and manual testing procedures:

- Use standards, everywhere, all the time. No exceptions to this rule.
- Have your website tested by human beings, particularly to find spelling or grammar errors, and to verify that the websites does what it is supposed to do, in all the target browsers and operating systems.
- Use JsUnit (http://www.jsunit.net/) to unit test your JavaScript code. Run the unit tests as often as possible, usually every night.
- Use JsDoc Toolkit (http://www.jsdoctoolkit.org/) to document your code and make this documentation available to the team.
- Compress your JavaScript files using tools like Dojo ShrinkSafe (http://alex.dojotoolkit.org/shrinksafe/) to make files lighter and speed up their execution.
- Use Selenium (http://www.openqa.org/selenium/) for browser compatibility and functional testing.
- Use online validators, particularly those of the World Wide Web consortium (http://validator.w3.org/). Use standards and fix your code to eliminate warnings and errors in the validation process.
- Have your team use several browsers in their development workflow (all those that are specified as mandatory in the specs); every new feature means a unit test, a functional test, and a "smoke test" reloading the page on the browser.
- Have the developers use tools like FireBug (http://www.getfirebug.com/[3]) and the Web Development Extension for Firefox (http://chrispederick.com/work/web-developer/).

Conclusions

Web development is a fascinating and constantly evolving activity. We are reaching a point where the technologies are getting more and more stable, secure and affordable, but I think that the web industry lacks of comprehensive and reliable integrated solutions yet.

References

Crockford, D.; "JavaScript: The World's Most Misunderstood Programming Language", [Internet] http://javascript.crockford.com/javascript.html (Accessed May 12th, 2007)

Wolter, J.; "Javascript Madness: Layout Engines", February 2007 [Internet] http://www.unixpapa.com/js/gecko.html (Accessed May 12th, 2007)

Zeldman, J.; "Fix Your Site With the Right DOCTYPE!", [Internet] http://alistapart.com/stories/doctype/ (Accessed May 12th, 2007)

[3] https://web.archive.org/web/20091115094010/http://getfirebug.com/

Update, September 10th, 2007: I am not the only one with this opinion.[4]

[4] http://www.ibm.com/developerworks/web/library/wa-cranky76/?ca=dgr-btw01BetterWebpages

Javascript Tips and Tricks (1)

Adrian Kosmaczewski

2007-08-03

As I said yesterday[1], JavaScript is the world's most misunderstood language[2], which means that you must unlearn what you have learned[3]. However complicated it might seem at first, it is quite easy to write and understand the most complex of JavaScript codes with just some examples.

Just a few observations before starting:

- Semicolons (;) are not mandatory, but **strongly recommended!**
- You can create strings using either 'apostrophes' or "quotes". "You can also 'mix them' as you want", but 'always keeping the "order" when using them'.
- Always use the "var" keyword when defining variables. Otherwise, the variables will be created on the "Global Object" of JavaScript, and this is a bad thing(TM): variables created in the "Global Object" **do not get garbage collected!**

This page[4] provides an excellent complement of information to know JavaScript better, as well as the Wikipedia page[5].

Some Tools

To work properly while programming in JavaScript, it is strongly recommended to use the following tools:

- Firefox[6]
- Firebug[7]
- Web Developer Extension[8]
- Internet Explorer Developer Toolbar[9]

[1] /blog/web-development-is-software-development
[2] http://www.crockford.com/javascript/javascript.html
[3] http://blogs.starwars.com/tomservo1976/59
[4] http://developer.mozilla.org/en/docs/A_re-introduction_to_JavaScript
[5] http://en.wikipedia.org/wiki/JavaScript
[6] http://getfirefox.com/
[7] https://web.archive.org/web/20091115094010/http://getfirebug.com/
[8] http://chrispederick.com/work/web-developer/
[9] http://www.microsoft.com/downloads/details.aspx?familyid=e59c3964-672d-4511-bb3e-2d5e1db91038&displaylang=en

The last one is just if you **must** Internet Exploder, which is a pain in the *** anyway.

Object “Literals”

First of all, every object in JavaScript is a map, and you can access properties and methods using either the **dot.syntax** or the **[“array”]** syntax:

```
var obj = {
    age: 42,
    "first and last name": "John Smith",     // yes, you can do that
    address: {
        street: "32 Kingston St.",
        city: "Springfield",
        zip: 12345
    },
    greet: function() {
        alert('hello! my name is ' + this["first and last name"] +
                ' and my age is ' + this.age.toString());
    }
};

// shows "John Smith"
alert(obj["first and last name"]);

// shows "true" ('===' is the identity operator)
alert(obj.age === obj["age"]);

// shows "object"
alert(typeof obj);
obj.greet();
```

Since the **dot.syntax** and the **[“array”]** syntaxes are equivalent, you must by now imagine that every “dot” in your code means a search into a dictionary (or literal object). **So, the less “dots” you use when calling an object, function or expression, the faster it is!**

To achieve this, use shortcuts:

```
var a = {
    very: {
        interesting: {
            JavaScript: {
                object: {
                    reference: "just an example!"
                }
            }
        }
    }
};
```

```
var shortcut = a.very.interesting.JavaScript.object.reference;
alert(shortcut);
```

JavaScript Base Classes

The following are the 7 core JavaScript classes that are part of the ECMA standard:

- `Object` (root class, like in Java)
- `String`
- `Number`
- `Array`
- `Date`
- `RegExp`
- `Function`

For a handy reference of the core JavaScript API, download and print this "cheat sheet"[10] (local copy here: JavaScript cheat sheet[11]).

It is important to know that, since JavaScript is a dynamic language, and that Functions are first-class objects, you can **add methods to a class on the fly**:

```
String.prototype.doSomething = function() {
    alert(this);
}

"hello!".doSomething();
```

The classes enumerated above are always present, in all JavaScript implementations (Adobe Flash ActionScript, Microsoft JScript, ECMAScript, etc).

When JavaScript runs in a web browser, other classes and objects are added, like Window, Document and others. These classes are part of the **DOM** (Document Object Model) and are browser specific (and based on W3C standards).

Functions

Functions are the basic block in JavaScript. You use them everywhere, you can pass them as parameters, attach them as event handlers, override them, delete them, etc. They can be anonymous or not:

```
function nonAnonymous() {
    alert('non anonymous!');
}

var nonAnonymousToo = function() {
    alert('non anonymous too!');
}
```

[10] http://www.ilovejackdaniels.com/javascript_cheat_sheet.pdf
[11] javascript_cheat_sheet.pdf

```
// This is the syntax for adding event handlers in Ext
field.on("click", function() {
    alert('now this is an anonymous function!');
});
```

You can also nest functions into functions, which is what they call **closures** in Lisp and other functional languages. Internal functions can access the variables created in the stack of their "parent" function:

```
function example() {
    var privateVar = 'a private var';

    function execute() {
        function go() {
            alert(privateVar);
        }
        go();
    }
    execute();
}

example();
```

Javascript Tips and Tricks (2)

Adrian Kosmaczewski

2007-08-04

Object-Oriented Programming in JavaScript

Functions are also used to represent **classes** when doing object-oriented JavaScript. There are several possible ways to write object-oriented JavaScript code, but they all turn around the concept of the `Function` class:

```
function Thing() {
    var privateField = "PRIVATE";

    // See below for more explanations about "this"   :)
    var self = this;

    var privateMethod = function() {
        alert('Private methods can be called from public methods');
        self.anotherPublicMethod();
    }

    this.publicField = "PUBLIC";

    this.publicMethod = function() {
        privateMethod();
        alert('From the public method;\nthis is a public value: '
                    + this.publicField
                    + '\nand this is a private value: '
                    + privateField);
    };

    this.anotherPublicMethod = function() {
        alert('You need a trick to call this
                        from a private method!');
    };
}

// Creating a new instance of "Thing"
var thingy = new Thing();
thingy.publicMethod();
// you can also call thingy["publicMethod"]();
```

As you can see, “methods” are just function objects attached as any other property. You can attach any other function to this property, changing the behavior of your class on the fly.

The “self” trick

As you can see in the above code, there is a variable called “self” (it could have any name) that is equal to “this”. This is a trick that allows private methods to access public methods, and you will see it in many JavaScript libraries. The problem can be summarized as follows: you cannot write the following

```
var privateMethod = function() {
    alert('Private methods can be called from public methods');
    this.anotherPublicMethod();
}
```

because `this` in that context means the `privateMethod()` function. What we want is the “this” that “points to” the current “Thing()” instance. Yes, it’s kinda complex, but it works perfectly well, because the `privateMethod()` function is a closure, and can access the stack variables of the `Thing()` function. Since `self` points to the right object, you can now call the public method that you want.

In summary, “this” points always to the immediately containing “function” object where you are located.

More ways to do the same thing

Another way to write the above code would be like this, but it has the drawback that you cannot access the “private” members, since you are attaching the public members to the “prototype” of the function, and as such, you are outside of the main context of the function Thing():

```
function Thing() {
    var privateField = "PRIVATE";

    var privateMethod = function() {
        alert('Private methods can be called from public methods');
    }
}

Thing.prototype.publicField = "PUBLIC";

Thing.prototype.publicMethod = function() {
    // privateMethod(); cannot be called here!
    // We're outside of the "Thing()" context
    alert('From the public method;\nthis is a public value: '
            + this.publicField
            + '\nbut you cannot access a private value!');
```

```
};

// Creating a new instance of "Thing"
var thingy = new Thing();
thingy.publicMethod();

// You can also call the public method as follows:
thingy["publicMethod"]();
```

The above syntax can also be written as follows:

```
function Thing() {
    var privateField = "PRIVATE";

    var privateMethod = function() {
        alert('Whatever');
    }
}

function Thing_publicMethod() {
    alert('Implementation');
};

function Thing_anotherPublicMethod() {
    alert('More implementation');
};

// "Header file" with the interfaces, all together
Thing.prototype.publicMethod = Thing_publicMethod;
Thing.prototype.anotherPublicMethod = Thing_anotherPublicMethod;

// Creating a new instance of "Thing"
var thingy = new Thing();
thingy.publicMethod();
```

which makes all the public methods appear together, like in a good old C or C++ interface header file.

The “Ext” way to do OO

Finally, nearly all Ext classes are written this way:

```
function Thing() {
    var privateField = "PRIVATE";

    var privateMethod = function() {
        alert('Private methods can be called from public methods');
    }

    return {
        publicField: "PUBLIC",
```

```
        publicMethod: function() {
            privateMethod();
            alert('From the public method;
                        \nthis is a public value: '
                        + this.publicField
                        + '\nand this is a private value: '
                        + privateField);
        }
    };
}

// Creating a new instance of "Thing"
var thingy = new Thing();
thingy.publicMethod();
```

In this last way of doing things, we're encapsulating the public interface of the class inside the "return" statement of the class, returning a dictionary (or literal) of members (fields and methods). This creates a neat separation of the public and private parts, with the neat advantage of allowing access to the private fields.

To use this last writing style, remember two things:

- Always remember to separate every member in the "return" clause with **commas**.
- Always remember to put the "return" and the **opening curly bracket in the same line!** That is, you **must** write "return {", otherwise, since semicolons are optional, the **function will return null!**

Inheritance

For inheritance, you can use the "prototype" keyword (basically the JavaScript native inheritance functionality) but you can use the Ext framework inheritance functions, which I recommend.

Javascript Tips and Tricks (3)

Adrian Kosmaczewski

2007-08-05

How to organize code in “namespaces”

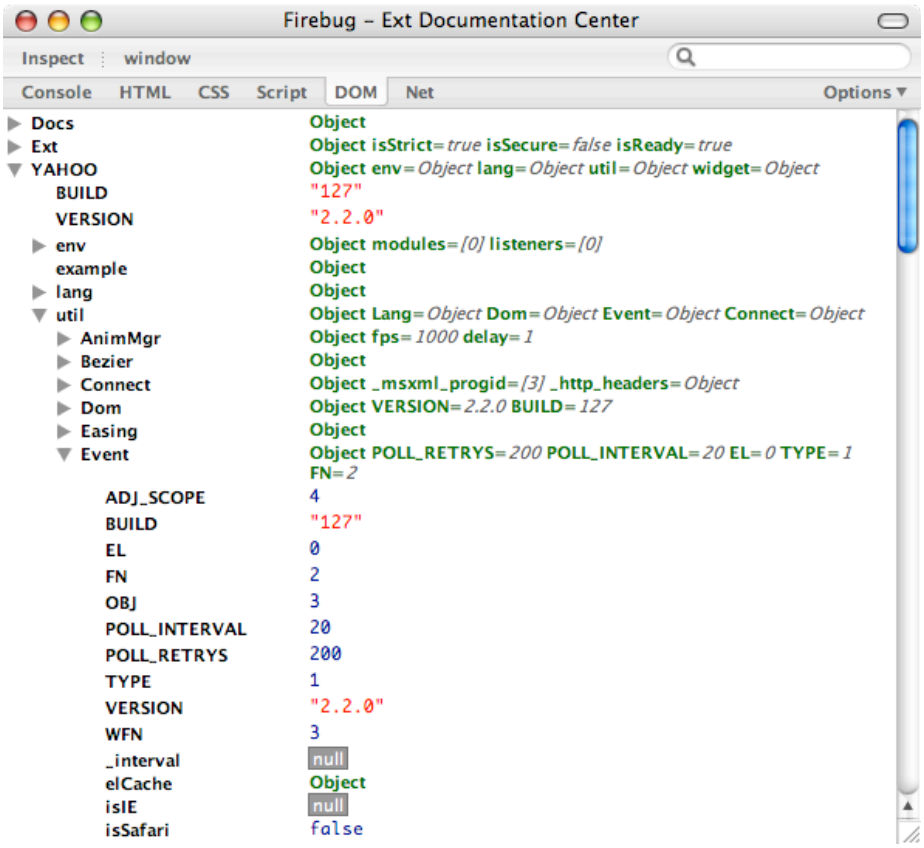
When you use lots of libraries in your code, you can easily pick up a function name that corresponds to a pre-existing name in some library that you might have included. To avoid that, you should create namespaces that encapsulate the code of your application:

```
var net = {
    kosmaczewski: {
        adrian: {
            blog: {
                articles: {},
                images: {},
                snippents: {},
                tutorials: {},
                rants: {}
            }
        }
    }
};

// Shortcut (for performance purposes)
var blog = net.kosmaczewski.adrian.blog;
```

Then, you can start adding members (functions, classes, variables, etc) to that namespace:

```
blog.rants.NewRant = function() {
    this.whatever = 'value';
    // code here...
}

blog.images.takePhoto = function() {
    return 'a nice picture';
}

blog.articles.numberOfPosts = 700;
```

The following image shows how Firebug[1] displays the objects added in their own namespaces, and also how the Ext and the Yahoo! code appear in their own namespaces:



Create Objects and Arrays the Easy Way

To create objects and arrays in JavaScript, you can of course use the constructor+methods syntax:

```
var obj = new Object();
var arr = new Array();

obj.prop = "value";
obj.method = function() {
    alert('method');
};

arr.push(23);
arr.push("yeah");
arr.push(obj);
```

[1] https://web.archive.org/web/20091115094010/http://getfirebug.com/

```
arr[2].method();
alert(arr);
```

While the above syntax is OK, many JavaScript interpreters can handle the following, completely 100% equivalent version, much faster and more “JavaScript-like” (by this I mean that you are more likely to find this in JavaScript libraries):

```
var obj = {
    prop: "value",
    method: function() {
        alert('method');
    }
};
var arr = [23, "yeah", obj];

arr[2].method();
alert(arr);
```

Create a Singleton Object

If you need to create a “singleton”, yet complex object, do not fall in the “classical” way of doing things: **do not create a class and then instantiate just one instance!**. Since JavaScript objects can be created on the fly, you can use object literals for that.

However, if you need to create just one object, with a complex structure, you can use the following trick:

```
var Singleton = function() {
    var privateValue = "private value";

    return {
        prop: "value",

        method: function() {
            alert(privateValue);
        }
    };
}();

Singleton.method();
```

The trick is in the line 11 of the example above:

```
}();
```

Do you see the parentheses after the closing bracket and before the semicolon? Well, this triggers the execution of the function, which “returns” a literal object with methods and properties, and which can reference private members (since they are closures).

This pattern is very common in the Ext Framework!

Scheduling Function Execution

Adding a method to the "Function" class, you can schedule its execution a couple of milliseconds in the future, encapsulating the `Window.setTimeout()` method:

```
Function.prototype.schedule = function(msec) {
    this.timeout = setTimeout(this, msec);
}

Function.prototype.cancelSchedule = function() {
    clearTimeout(this.timeout);
}

function doSomething() {
    alert('doSomething');
}

doSomething.schedule(5000);
```

The Ext framework has a method that does exactly this, called `defer()`. Very handy!

Concatenating Strings

If you have to concatenate strings, avoid using the + operator whenever possible; the Array class can be used as a Java `StringBuffer` or a .NET `StringBuilder`, as follows:

```
var s = ["a", "long", "array", "of", "strings"];
s.push("is");
s.push("here");
document.write(s.join("<br>"));
```

The above code will display the following output on the web page:

```
a
long
array
of
strings
is
here
```

The use of the "join()" method is considered as slightly faster than using the "+" operator, which implies much more object copies in memory, which in turn triggers the garbage collector much more often.

Iterating over Arrays

When you operate on array objects, you usually end up writing code like this:

```
function operate(obj) {
    alert(obj);
}

var arr = [54, 25, 68];
for (var i = 0; i < arr.length; ++i) {
    operate(arr[i]);
}
```

But things do not have to be that awful all the time:

```
Array.prototype.each = function(func) {
    for (var i = 0; i < this.length; ++i) {
        func(this[i]);
    }
}

function operate(obj) {
    alert(obj);
}

var arr = [54, 25, 68];
arr.each(operate);
```

Now you have code that's much easier to read!

Using `toString()` for Reflection

Let's suppose that you have the code written above

```
function Thing() {
    var privateField = "PRIVATE";

    var privateMethod = function() {
        alert('Private method');
    }

    return {
        publicField: "PUBLIC",

        publicMethod: function() {
            alert('Public method');
        }
    };
}

// Creating a new instance of "Thing"
```

```
var thingy = new Thing();

// You want to see what's inside, right?
alert(thingy);
```

The last instruction above shows a laconic `[object Object]` which does not tell much about what's inside your object... Actually, the `alert()` function, when applied to any JavaScript object, will call the "toString()" method to its parameter: so try adding a `toString()` to your objects instead:

```
function Thing() {
    var privateField = "PRIVATE";
    var self = this;

    var privateMethod = function() {
        alert('Private method');
    }

    return {
        publicField: "PUBLIC",

        publicMethod: function() {
            alert('Public method');
        },

        toString: function() {
            var re = /function (.*)\(\) {/g;
            var a = self.constructor.toString().split("\n")[0];
            var cls = "Class " + re.exec(a)[1];
            var s = [cls, "", "Public API:"];
            for (var item in this) {
                s.push(item);
            }
            return s.join("\n");
        }
    };
}

// Creating a new instance of "Thing"
var thingy = new Thing();

// You'll get a very rudimentary reflection output
alert(thingy);
```

With this code, you'll get this output in a dialog box:

```
Class Thing

Public API:
publicField
publicMethod
```

Easy Code Injection

This code allows you to inject arbitrary code around any function, like if you were doing some cheapo AOP:

```
Function.prototype.wrap = function(before, after) {
    before();
    this();
    after();
};

function doBefore() {
    alert('do before');
}

function doAfter() {
    alert('do after');
}

function test() {
    alert('inside test');
}

// test();
test.wrap(doBefore, doAfter);
```

Do not hesitate to submit your own snippets or tricks! I hope this is useful to you.

Javascript Tips and Tricks (4)

Adrian Kosmaczewski

2007-08-06

For the last article of this series, I'll let Douglas Crockford do the talk :) This is an amazing video about the good and the bad parts of JavaScript, as seen by a guy that seriously, seriously knows his stuff: Douglas works in the YUI team at Yahoo!, and has written a lot about JavaScript: he coined the phrase about JavaScript being the world's most misunderstood language, and wrote the amazing JSLint tool.

I the video below, Douglas exposes all his thoughts about the language, both the good and the evil, all out of his own experience as a guy who had to rediscover JavaScript in 2000. Really interesting stuff!

The Explosion of the Columbia Space Shuttle

Adrian Kosmaczewski

2007-08-10

The explosion of the Columbia Space Shuttle uncovered several internal problems in the National Aeronautics and Space Administration (NASA). This paper will summarize the reasons of the accident, highlighting the organizational issues, and provide an overview of the roles played by the different stakeholders of the project.

[1]

The Columbia Accident

The Columbia Space Shuttle, in its mission “STS-107”, exploded over Texas on February 1st, 2003, killing all of its seven crew members. Its tremendous explosion, seen and heard from the ground while the shuttle was on its way to Florida, led to the complete interruption of the USA space program. The Columbia Accident Investigation Board (CAIB) was formed immediately after the event to provide much needed answers, such as the one to the question of the origin of the

[1] https://www.flickr.com/photos/bootbearwdc/33188675/

accident.

The CAIB issued a report in August 2003, freely distributed on the CAIB website[2], that provides a twofold answer to the explosion:

1. A technical answer: "The physical cause of the loss of Columbia and its crew was a breach in the Thermal Protection System on the leading edge of the left wing." (CAIB, "Report Volume I", page 49)
2. An organizational answer: "The Board found that there is a"broken safety culture" at NASA (pp. 184-189). Schedule pressure (pp. 131-139) related to construction of the International Space Station, budget constraints (pp. 102-105), and workforce reductions (pp. 106-110) also were factors. The Board concluded that the shuttle program "has operated in a challenging and often turbulent environment...." (p. 118) "It is to the credit of Space Shuttle managers and the Shuttle workforce that the vehicle was able to achieve its program objectives for as long as it did." (p. 119)"

(Marcia S. Smith, "Synopsis", page 2; the page numbers reference CAIB, "Report Volume I").

In pages 192 and 193 of the Report, the CAIB enumerates its findings from the perspective of the organizational point of view, and proposes key modifications.

It is interesting to see how CAIB's Report stresses the organizational answer for the explosion of the Columbia Space Shuttle, and this is particularly obvious in chapter 8 (pages 195 to 204) where the Report compares the tragedy of the Columbia to that of the Challenger, in 1986.

The impact of such an accident is clearly put into light when enumerating the major stakeholders of USA's space program:

1. The USA Government: Both the White House and the Congress are deeply influenced by the success or failure of the space program, but in different ways:
    1. There is a great sense of pride of the American people regarding the space program; as such, the popularity of the President of the United States has historically been greatly influenced by its results; the desire and objective of the White House is then to have a greater, better space program each year (see for example this link[3])
    2. The Congress has the hard task of deciding the amount of money that will be given to NASA every year, and historically, the budget for the space program has been eroded sinde the mid seventies, after the Apollo 17 mission, the last

[2] http://caib.nasa.gov/

[3] http://www.whitehouse.gov/news/releases/2004/01/20040114-3.html

one sent to the Moon (Source[4]). The position of the Congress is then to try to reduce this amount of money, every year, to compensate with the enormous public debt of 64.7% of GDP (CIA World Factbook, 2006), which clearly demands a reduction of public expenses. This is currently provoking much turmoil in the NASA scientist community (see for example this link[5])

2. NASA's staff: CAIB's Report clearly identifies two different kinds of stakeholders; on one side the upper management, and on the other the technical teams (engineers, astronauts, scientists in general). Both have different and, lately, opposed objectives:

    1. The upper management targets recognition and political success (the NASA is an extremely big budget with high visibility) while they not always have the technical skills needed to understand the day-to-day jobs of their reportees. At the same time, facing a reduction of budget, they have to fight against the technical staffs to reduce their size, salaries or otherwise overall capacity, leading to internal struggles:

    > The subcommittee asked that we discuss the four major management challenges we identified at NASA in our latest Performance and Accountability Series report. These include: (1) strengthening human capital; (2) controlling International Space Station costs; (3) implementing a faster, better, cheaper approach to space exploration; and (4) correcting weaknesses in contract management.

(United States General Accounting Office, 2002)

1. On the other side, the engineering staff directs its actions towards the achieve

    1. The public: There is great interest in the public to know the results of the space program, to follow its discoveries, and this somehow "feeds" the pride that the American people has about its space program (by far the biggest on Earth). Furthermore, NASA being a public entity, funded by taxes, the public has a great interest in knowing how the funds are spent and invested.

[4] http://history.nasa.gov/pocketstats/sect%20D/NASA%20Budget.pdf
[5] http://www.msnbc.msn.com/id/11642092/

 [6]

Conclusion

The organizational problems of the NASA are due to the extreme complexity of the space program (by far the most complex project ever undertaken), and by the diversity of nature and interests of the stakeholders from inside and outside the organization. The (second) explosion of the space shuttle is a warning sign calling for a major internal restructuration.

[6] https://www.flickr.com/photos/royalty-free-images/146116117/

This extreme example shows clearly how the internal structure of an organization can greatly impact a project, and how the stakeholders must interact and behave in order to help the project. I can only recommend reading the CAIB Report, since it provides a deep understanding of the current situation of the space program.

Finally, I would like to point out that while this is not an IT specific example, it constitutes a clear and extreme one of how organizational issues can affect the outcome of a project. Nevertheless, it must be said that the NASA has historically been one of the major software organizations; its Software Engineering Laboratory (SEL) has the mission

> to lead and facilitate the improvement of software engineering practices in the Information Systems Center (ISC), within Goddard and NASA, and in the wider software development industry

(NASA SEL, 2006).

References

Columbia Accident Investigation Board (CAIB) Website [Internet], http://caib.nasa.gov/ (Accessed April 12th, 2006)

CAIB, Report Volume I, August 2003 [Internet], http://caib.nasa.gov/news/report/pdf/vol1/full/caib_report_volume1.pdf (Accessed April 12th, 2006)

CIA World Factbook, "USA", [Internet] http://www.cia.gov/cia/publications/factbook/geos/us.html (Accessed April 12th, 2006)

Marcia S. Smith, Congressional Research Service, Library of Congress, "NASA's Space Shuttle Columbia: Synopsis of the Report of the Columbia Accident Investigation Board", September 2nd, 2003 [Internet], http://www.house.gov/science/hearings/full03/sep04/crssynop.pdf (Accessed April 12th, 2006)

NASA History, "NASA's Budget Authority in 1996 Dollars" [Internet], http://history.nasa.gov/pocketstats/sect%20D/NASA%20Budget.pdf (Accessed April 12th, 2006)

NASA SEL (Software Engineering Laboratory), "SEL Vision and Mission" [Internet] http://sel.gsfc.nasa.gov/website/welcome/vision-mission.htm (Accessed April 12th, 2006)

United States General Accounting Office, "NASA MANAGEMENT CHALLENGES; Human Capital and Other Critical Areas Need to be Addressed", July 18th, 2002 [Internet], http://www.gao.gov/new.items/d02945t.pdf (Accessed April 12th, 2006)

About Mercury TestDirector for Quality Center

Adrian Kosmaczewski

2007-08-13

Mercury TestDirector for Quality Center is an enterprise-class solution, recognized as one of the most complete and integrated software suites geared towards the management of testing- and quality-related activities. Mercury Interactive, the company behind TestDirector, was bought by Hewlett Packard in 2006 for USD 4.5 billion (Wikipedia). TestDirector is used by several important organizations such as Nextel, the United States Navy, Nestlé, Qualcomm and Shell (Mercury Website, 2007a).

Mercury TestDirector provides the following features (Mercury Website 2007b, 2007c & 2007d):

- **Requirements Management:** analysts can use TestDirector to define the complete business needs of the software solution to be tested and evaluated, as a list of individual pieces of functionality; this way, the testing teams can evaluate the testing coverage across the software solution in terms of covered functionality.
- **Test Planning:** test managers can use TestDirector to build, document and share the complete test plan, even allowing the direct import of Microsoft Office documents into the TestDirector repository.
- **Test Automatization and Scheduling:** test engineers can create integrated test scripts directly in TestDirector, and trigger their execution at defined times.
- **Defect Management:** developers are notified of defects found in the system using the integrated bug database.
- **Reporting:** TestDirector is able to export data to a wide variety of formats and supports, to allow stakeholders to be aware of the current state of the quality processes.
- **Integration with External Tools:** TestDirector can be integrated through its open, documented API with external bug tracking systems, such as Atlassian JIRA (JIRA Community Space) or source code management systems, such as Subversion (Collab.net Website) or IBM Relational ClearCase (Mercury Website, 2007e) among other tools.
- **Multi-platform Support:** Mercury TestDirector is a J2EE appli-

cation, that can be installed in any system with a JRE (Windows, Linux, Solaris) and a compatible application server (JBoss, WebSphere, WebLogic), using many different database backend systems (Oracle or MS SQL Server).

TestDirector provides the main benefit of the complete integration of the whole quality process, thanks to its complete toolkit:

> Using TestDirector for Quality Center, multiple groups throughout your organization can contribute to the quality process:
>
> - Business analysts define application requirements and testing objectives.
> - Test managers and project leads design test plans and develop > test cases.
> - Test managers automatically create QA testing requirements and test assets for SOA services and environments.
> - Test automation engineers create automated scripts and store them in the repository.
> - QA testers run manual and automated tests, report execution results, and enter defects.
> - Developers review and fix defects logged into the database.
> - Project managers can export test and resource data in various reports, or in native Microsoft Excel for analysis.
> - Product managers decide whether an application is ready to be released.
> - QA analysts can auto-generate test asset documentation in Microsoft Word format.

(Mercury Website, 2007a)

In conclusion, TestDirector enables an unparalleled level of integration between the testing and development teams, while at the same time offering a high degree of visibility to all stakeholders about the quality process.

References

Collab.net Website, “Supporting Mercury Test Director bug tracking with Subversion”, [Internet] http://connectors.open.collab.net/alm-process/2-Articles%20and%20docs/documents/td_svn_use_case.pdf (Accessed May 5th, 2007)

JIRA Community Space, “JIRA to Mercury TestDirector bridge”, [Internet] http://confluence.atlassian.com/display/JIRACOM/JIRA+to+Mercury+TestDirector+bridge (Accessed May 5th, 2007)

Mercury Website, “Mercury TestDirector”, 2007a [Internet] http://www.mercury.com/us/products/quality-center/testdirector/ (Accessed May 5th, 2007)

Mercury Website, “Mercury TestDirector - How it Works”, 2007b [Internet] http://www.mercury.com/us/products/quality-center/testdirector/works.html (Accessed May 5th, 2007)

Mercury Website, “Mercury TestDirector - System Requirements”, 2007c [Internet] http://www.mercury.com/us/products/quality-center/testdirector/requirements.html (Accessed May 5th, 2007)

Mercury Website, “Mercury TestDirector - Data Sheet”, 2007d [Internet] http://www.mercury.com/us/pdf/products/datasheets/DS-1134-0906-testdirector.pdf (Accessed May 5th, 2007)

Mercury Website, “Rational ClearCase Version Control Add-in”, 2007e [Internet] http://updates.merc-int.com/testdirector/td80/version_control/RCC_vc/index.html (Accessed May 5th, 2007)

Wikipedia, “Mercury Interactive”, [Internet] http://en.wikipedia.org/wiki/Mercury_Interactive (Accessed May 5th, 2007)

Browsers' Multiple Connection Settings

Adrian Kosmaczewski

2007-08-16

I'm not such a "power user" when it comes to web browsing, and having good connectivity both at home and at work helps forgetting about tweaking the maximum number of TCP connections in my web browser. However, I can think of a two cases where the establishment of multiple connections might bring great value to the power user: first, **multiple file downloads**. Being able to download several files at once might be really helpful in some situations. Secondly, **in the case of complex web pages**. When web pages contain different elements, the required download time goes up (which is often the case: images, sound, video, Java applets, Flash animations, JavaScript files, stylesheets, etc., are ubiquitous nowadays).

In this article I will provide an overview of multiple connection availability in some Mac browsers, and also provide some tips for Internet Explorer for Windows, in the context of the HTTP RFC.

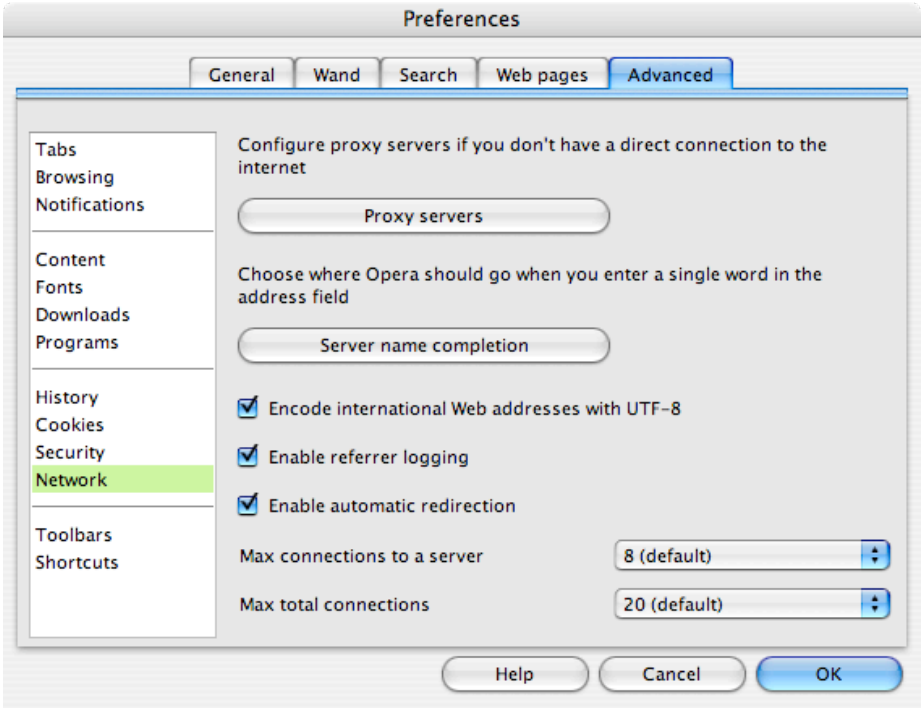
Browsers on Mac OS X

First I took a look into the browsers that I have installed at my home machine; I do some web design in my spare time, that's why I keep a high number of them, to test the standard-compliance of my designs, which is like using several compilers for testing your C++ source code:

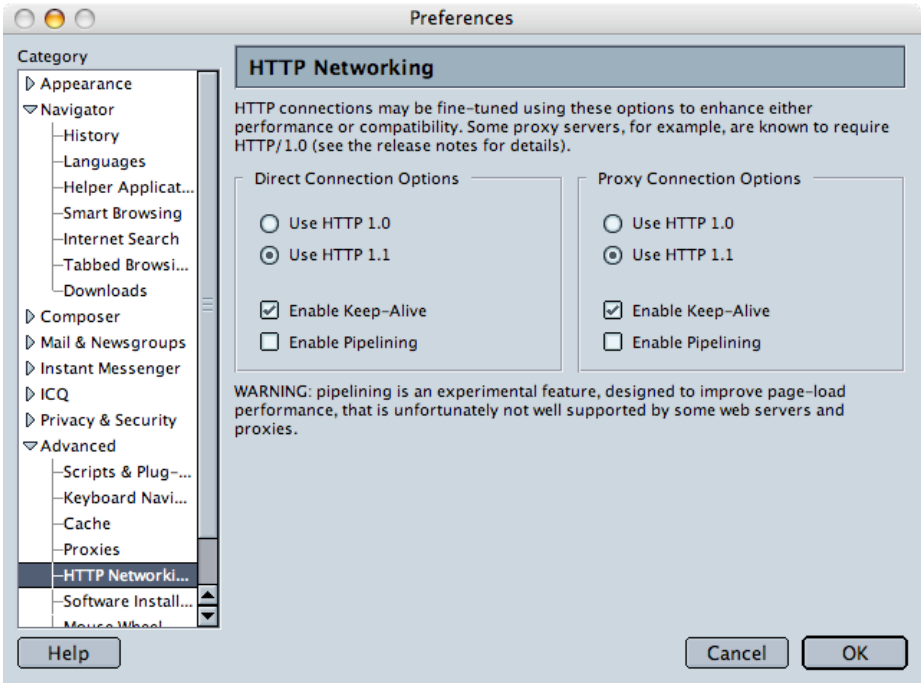
- Camino[1]
- Apple Safari[2]
- Mozilla Firefox[3]
- Opera[4]
- Shiira[5]
- OmniWeb[6]
- Netscape 9[7]

[1] http://www.caminobrowser.org/
[2] http://www.apple.com/macosx/features/safari/
[3] http://www.mozilla.com/en-US/firefox/
[4] http://www.opera.com/
[5] http://shiira.jp/en
[6] http://www.omnigroup.com/applications/omniweb/
[7] http://browser.netscape.com/

In this case, only Opera directly provided the capacity to tweak the number of connections:



As shown above, Opera 9 (at least for the Mac, but I'm sure that in other platforms too) allows users to change the number of connections, from 1 to 128. On the other side, Netscape version 7.2 (now obsolete after the recent release of version 9) allows you to change some other settings about the HTTP connection, namely "pipelining" and "keep-alive" connections:



No other browsers (from the list above) provide this capacity. It is not a real surprise that Mac browsers "hide" from the user the capability of changing such a setting, since the general guidelines for application development on this platform favor user-friendliness and accessibility, at the cost of expert settings.

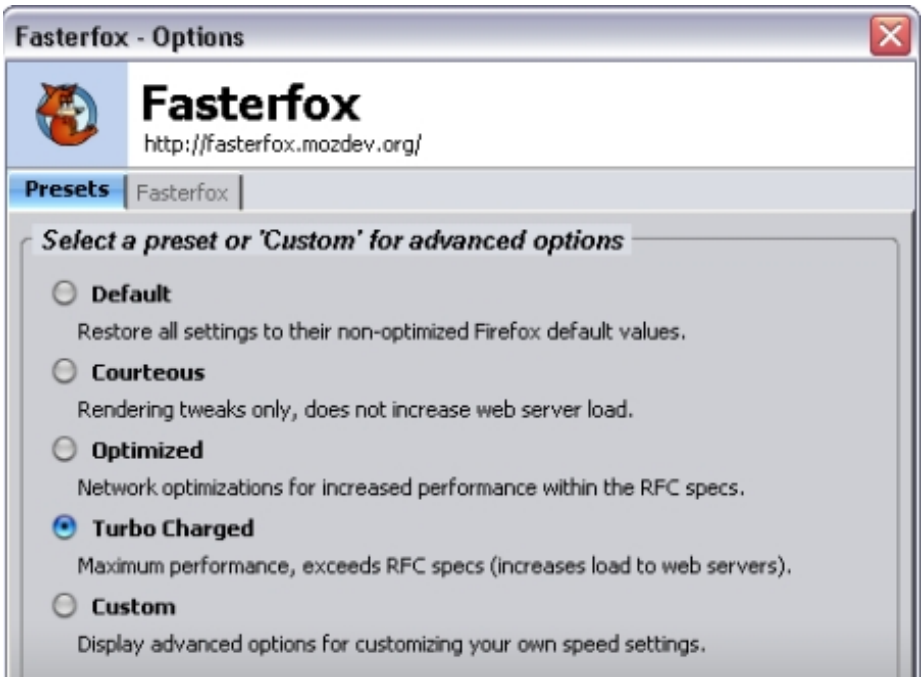
Finally, it is worth noting that external tools such as Yazsoft's Speed Download[8] allows Mac users to open simultaneous connections to a web server, in order to speed up huge downloads.

Mozilla Firefox

Mozilla Firefox users (on any operating system) can install the "Fasterfox" plugin[9] which allows to "tweak many network and rendering settings such as simultaneous connections, pipelining, cache, DNS cache, and initial paint delay."

[8] http://www.yazsoft.com/

[9] https://addons.mozilla.org/firefox/1269/



Internet Explorer for Windows

A quick search on the net brought interesting results. First of all, it seems that Internet Explorer does not provide a customization setting to change the number of connections, at least not directly; the maximum limit of connections is a property of WinInet, a Windows networking library used by all applications that wish to connect to the Internet. As such, power users not afraid of playing with the registry can change the values regarding the maximum number of connections to a server:

> HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Internet Settings MaxConnectionsPerServer REG_DWORD (Default 2) Sets the number of simultaneous requests to a single HTTP 1.1 Server MaxConnectionsPer1_0Server REG_DWORD (Default 4) Sets the number of simultaneous requests to a single HTTP 1.0 Server

(Source[10])

In the same document, it is stated that “WinInet limits connections to a single HTTP 1.0 server to four simultaneous connections. Connections to a single HTTP 1.1 server are limited to two simultaneous connections. The HTTP 1.1 specification (RFC2616) mandates the two-connection limit.”. However, I found out that in the official HTTP/1.1 documentation, namely the RFC 2616, the following paragraph states

[10] http://support.microsoft.com/kb/183110

the upper limit of 2 connections as a recommendation, rather than a "mandatory" setting:

> 8.1.4 Practical Considerations: (...) Clients that use persistent connections SHOULD limit the number of simultaneous connections that they maintain to a given server. A single-user client SHOULD NOT maintain more than 2 connections with any server or proxy. A proxy SHOULD use up to 2*N connections to another server or proxy, where N is the number of simultaneously active users. These guidelines are intended to improve HTTP response times and avoid congestion.

(Source[11])

[12]

The RFC document is extremely easy to read, and provides good reasons and explanations about why not to use a higher number of connections simultaneously:

- Not all web servers can cope with the workload produced by several open connections;
- It reduces the use of resources (bandwidth, memory, CPU time) in both clients, routers and servers;
- It helps improve the responsiveness and the evolution of the overall network.

Please refer to section 8.1.1 of the RFC, where some advantages of

[11] http://www.faqs.org/rfcs/rfc2616.html
[12] https://www.flickr.com/photos/missrogue/199208087/

persistent connections are described with great detail. Furthermore, this interesting blog entry on the IEBlog[13] provides more information about the reasons for the default setting.

Conclusion

The suggested reduction of the maximum number of connections seems to have been followed closely by browser developers; it is either impossible or very difficult to change this setting, with the exception of Opera, which makes is really simple. There are a number of long-term advantages to the limitation of the number of connections, and this has surely had a strong impact, helping the Internet to grow (and not to stall) during the past thirty years.

[13] http://blogs.msdn.com/ie/archive/2005/04/11/407189.aspx

Design by Contract

Adrian Kosmaczewski

2007-08-23

Even if Design by Contract is a trademark (Eiffel Software, 2007) the idea behind it is the more general one of “defensive programming”. As software developers, we often concentrate our efforts in the main code of the application, which is the interesting part, and that provides the realization of the use cases identified during the early phases of the project.

My opinion is that defensive programming techniques lead to more secure, stable, and less bug-prone code, and that they require less documentation, since the resulting code is more “self-documenting”. Moreover, I will describe in this paper language-independent techniques, that can be used in different situations and systems, that are somehow similar to the Eiffel approach.

Design by Contract

In a previous entry[1] in my blog, I have described the Ariane 5 rocket disaster of 1996, its cause and how the notion of defensive programming could have helped avoid it:

> Of course neither me nor my colleagues work on Eiffel (yet). But, we do create software, we do handle exceptions (do we?), and we do lose money, credibility and faith every time there’s an unhanded exception in our software. These exceptions cost us a lot: that “Design by Contract (TM)” thing can positively help us, just by rethinking the way we build our software. Here’s my idea: even if we don’t use Eiffel, our good-old Algol-related languages can be used in pretty much the same way as Eiffel behaves, but of course it requires some of our own brainy CPU time. The trade off is simply a much clear interface that fits into a higher level, architectural view of the system, explicitly stating the valid ranges of execution for our code, and helping out in setting unit and integration tests.

(Kosmaczewski.net, 2005)

[1] /blog/the-exception-to-the-rule/

Eiffel's idea is coherent with the concept of "Defensive Programming". It is not a new trend in programming, but if you Google for it you will find quite a few papers describing common techniques and best practices, like the one on CodeProject.com that I provide in the references (Manderson, 2004).

The idea behind defensive programming is that you should not trust what comes from outside your code, even if it is your own code that calls other pieces of your own code; you could just as well call it "limited applied paranoia". This is important not only for stability purposes (you do not want to call methods on a null pointer) but also for security; take for example the well known "SQL injection attacks" that happen when you trust too much the inputs of your users... a little verification helps to avoid disasters.

In Eiffel, this is done using the following syntax:

```
method_name (parameter_name: INTEGER): INTEGER is
require
    parameter_name <= some_maximum_value
    -- more conditions, if needed...
do
    -- code of the method here
ensure
    -- postconditions that must always be met
    -- no matter what happens, here
end
```

(Source: Kosmaczewski.net, 2005)

Of course, not all programming languages provide this kind of pre- and post-verification syntax; some ways to apply defensive programming in non-Eiffel languages, could be for example:

- Asserting for validity
- Using Aspect-Oriented Programming
- Using proper exception handling

I will give a short overview of these in the following paragraphs, comparing them with the Eiffel approach.

Asserting

"Asserting" is a common technique used in C and C++, but that can be used as well in any other language; basically it consists in using a macro or function that will raise an exception (or halt the program execution altogether) if a condition is true. Typically this is done before sending a message to a null pointer, since this mistake is a common one:

> The ANSI assert macro is typically used to identify logic errors during program development by implementing the expression argument to evaluate to false only when the pro-

> gram is operating incorrectly. After debugging is complete, assertion checking can be turned off without modifying the source file by defining the identifier NDEBUG. NDEBUG can be defined with a /D command-line option or with a #define directive. If NDEBUG is defined with #define, the directive must appear before Assert.h is included.

(MSDN, 2007)

The use of macros help to remove the asserts from the shipping code, that are usually only used during development and debugging. You do not want to ship code that discloses too much information about errors…

The situation in which asserting is useful is not uncommon in other languages; the much feared “null pointer exception” can happen in JavaScript, Ruby, C#, Java, and many other languages. In Eiffel, this would be handled in the “require” block, right before the main code execution.

Just for the record, the Apple Cocoa runtime (and in general the Objective-C language runtime) allows messages to be sent to “nil” objects (aka null pointers), without problem. However strange this might sound, this has an interesing side effect; even if the developers forget to initialize a pointer, and send a message to it, nothing will happen; the application will not crash, and this particular feat is one of the secrets of the stability of the overall Mac system. It is not that developers make less mistakes or that some magic applies; the Cocoa runtime proactively protects users from sloppy programmers, providing a more stable environment for them: the application might not do what the user want, but at least it does not crash.

Aspect-Oriented Programming

AOP can be used in this context as well, intercepting messages before and after methods execution, to verify their conformity and their correctness. The problem with AOP is that the definition of aspects is usually done in a different code file, which makes it harder to understand and maintain; this is where the Eiffel approach is the most interesting, since the “require” and “ensure” methods are somehow part of the method’s signature.

However, the advantage of AOP is that the definition of poincuts is much more flexible, and one could theoretically inject code in different situations, without having to touch the original code (and thus reducing coupling and cohesion); the AOP runtime would take care of the weaving for us. Moreover, the weaving could be changed at runtime, while “require” and “ensure” conditions in Eiffel are compiled and statically linked.

Just for the example, I will show a bit of Ruby on Rails code that shows a class that provides a “hook” for before-execution methods; these

methods are called automatically by Rails before any execution:

```
# The administration functions allow authorized users
# to add, delete, list, and edit products.(...)
#
# Only logged-in administrators can use the actions
# here. (...)

class AdminController < ApplicationController

    before_filter :authorize

    # List all current products.
    def list
    @product_pages, @products = paginate :product,
                                         :per_page => 10
    end

    # Show details of a particular product.
    def show
    @product = Product.find(@params[:id])
    end

    (...)

end
```

The important part of the code above is the `before_filter :authorize` line, that tells Ruby to automatically call the `authorize` method before executing the other methods. The `authorize` method belongs the to the `ApplicationController` class, and the `before_filter` hook is defined in the `ActionController::Base` class, that's part of the Ruby on Rails framework. Rails uses the dynamicity of Ruby to "detect" a call to a method, and intercepts it to inject the desired behavior before the execution of that method. Similar hooks exist for post processing.

It is interesting to note that such mechanisms can also be implemented in static, compiled languages using class hierarchies, where base class' methods call virtual methods of subclasses, similar to how ASP.NET processes pages.

Proper Exception Handling

Modern object-oriented runtimes, such as Java, .NET, Ruby and Cocoa use "exceptions" to handle errors. Exceptions are objects that are "thrown" whenever some unexpected condition is met during runtime to the method callers up in the stack, until some method "catches" it (hopefully someone will). This objects carry a whole meaning about the error, and are much more explicit than the C / C++ approach of returning numeric codes (HRESULTs, anyone?). By looking an excep-

tion, maintainers can see the context of execution of the code, and find the bugs that have created (or allowed) it to happen.

Even if the idea is interesting, it is also known that Exception handling is an expensive way to handle errors; for each exception thrown, runtimes have to walk the stack and look for possible handlers. This operation is expensive (Sintes, 2001) and that is why exceptions should not be used to control program flow.

The canonical example to show this is the following: instead of writing this (pseudo) code

```
try
{
    openFile(fileName);
}
catch (FileNotFoundException e)
{
    doSomething(e);
}
```

one could write the following, functionally similar, but more "defensive" code:

```
if (fileExists(fileName))
{
    openFile(fileName);
}
else
{
    notifyProblem();
}
```

If the above code is part of a performance-sensitive system, such as a web application, and the "file not found" situation happens more often than not, then a lot of processing power could be saved by just introducing this small change in the implementation.

In the case of Eiffel, the "contract" for the above "openFile" method would not only include the file name, but also a "require" block stating that the file must exist before any processing. This way, all calls to openFile would be safe, and as a result, clients would not need to check that fact before calling the method. Less code, cheaper to maintain, and more stable.

Conclusion

Defensive programming is a mind paradigm; you can apply them in any language, be it procedural of object-oriented, and provides stronger code, resistant to changes and self-documenting.

References

Eiffel Software, “Building bug-free O-O software: An introduction to Design by Contract(TM)”, [Internet] http://archive.eiffel.com/doc/manuals/technology/contract/ (Accessed June 22th, 2007)

Kosmaczewski, Adrian, “The Exception to the Rule”, February 15th, 2005 [Internet] /blog/the-exception-to-the-rule/ (Accessed June 22th, 2007)

Manderson, Rob, “Defensive Programming”, August 6th, 2004, [Internet] http://www.codeproject.com/gen/design/defensiveprogramming.asp (Accessed June 22th, 2007)

MSDN, “assert (Visual C++ Libraries)”, [Internet] http://msdn.microsoft.com/library/en-us/vclib/html/_CRT_assert.asp?frame=true (Accessed June 22th, 2007)

Sintes, Tony, “Does exception handling impair performance?” [Internet] http://www.javaworld.com/javaworld/javaqa/2001-07/04-qa-0727-try.html (Accessed June 22th, 2007)

Thomas, Dave & Heinemeier Hansson, David, “Agile Web Development with Rails”, The Pragmatic Programmers, 2005, ISBN 0-9766940-0-X, sample source code taken from http://pragmaticprogrammer.com/titles/rails1/code.html (Accessed June 22th, 2007)

How to Test Software Security?

Adrian Kosmaczewski

2007-08-27

Howard and LeBlanc give a very complete answer to this question in their classic “Writing Secure Code” book:

> Most testing is about proving that some feature works as specified in the functional specifications. If the feature deviates from its specification, a bug is filed, the bug is usually fixed, and the updated feature is retested. Testing security is often about checking that some feature appears to fail. What I mean is this: security testing involves demonstrating that the tester cannot spoof another user’s identity, that the tester cannot tamper with data, that enough evidence is collected to help mitigate repudiation issues, that the tester cannot view data he should not have access to, that the tester cannot deny service to other users, and that the tester cannot gain more privileges through malicious use of the product. As you can see, most security testing is about proving that defensive mechanisms work correctly, rather than proving that feature functionality works. In fact, part of security testing is to make the application being tested perform more tasks than it was designed to do. Think about it: code has a security flaw when it fulfills the attacker’s request, and no application should carry out an attacker’s bidding.

(Howard & LeBlanc, 2003, page 568).

This impressive description is followed by a complete test plan for security, that can be summarized as follows:

1. Decompose the application in components: networking, user interface, database, etc.;
2. Identify the component’s interfaces: network sockets, dialog boxes, databases, the Windows’ registry, microphones, or even the clipboard! This will define the “attack surface” of the system;
3. Rank the above interfaces by potential vulnerability;
4. Identify the data accessed by each interface: network packets, environment variables, user input, command-line parameters, etc.;

5. Attack each interface with ad hoc techniques: network tools, badly-formed command line parameters, invalid XML data, long strings, malformed URLs, cross-site (XSS) scripts, etc. (Howard & LeBlanc, 2003, pages 570 to 613)

[1]

The framework proposed above shows clearly that security testing has a life of its own. "Classical" testing concentrates efforts in the functionality to be delivered by the final system, while security testing has an orthogonal set of goals, many of which are usually not considered part of the final deliverable, but might appear as fundamental later, when the application is deployed in the "real world".

I consider security testing actually as one aspect of the broader concept of secure design: **not taking security in the design process is a recipe for disaster.** Software applications that have been designed with security in mind are actually able to deliver secure features, are strong when they are attacked, and are able to recover from or report about those attacks. The canonical comparison is that of the BSD family of operating systems and Windows Server; in sensible environments, BSD-derived operating systems (like OpenBSD, Mac OS X or NetBSD) are considered as virtually invulnerable, given the strong focus on security that they have been given from the very beginning:

> OpenBSD believes in strong security. Our aspiration is to be NUMBER ONE in the industry for security (if we are not already there). (...)

[1] https://www.flickr.com/photos/nedrichards/56919158/

> Like many readers of the BUGTRAQ mailing list, we believe in full disclosure of security problems. (...)
>
> Our security auditing team typically has between six and twelve members who continue to search for and fix new security holes. We have been auditing since the summer of 1996. The process we follow to increase security is simply a comprehensive file-by-file analysis of every critical software component. (...) Code often gets audited multiple times, and by multiple people with different auditing skills.(...)
>
> Our proactive auditing process has really paid off. Statements like "This problem was fixed in OpenBSD about 6 months ago'' have become commonplace in security forums like BUGTRAQ.

(OpenBSD)

A search for "secure operating system"[2] in Google brings OpenBSD as the third result (as of May 17th, 2007)

> OpenBSD is often noted for its code auditing and integrated crypto, but the security features go far beyond this. OpenBSD was built from the ground up on the model of being a fabric woven with security in mind, not a patchwork of bug fixes and security updates. This has led to OpenBSD finally being recognized today for what it is: the most secure operating system on earth.

(ONLamp.com, 2000)

The most interesting statement in the OpenBSD citation above is

> Our security auditing team typically has between six and twelve members who continue to search for and fix new security holes.

This shows that the OpenBSD team takes security very seriously at all stages of the production of the system. **I think that more generally, when security is designed as a system feature rather than an afterthought, security testing cannot be distinguished from the overall testing effort, even if it consists of a special set of procedures.**

BSD is not dying, as this funny video[3] tries to convince us!

References

Howard, M.; LeBlanc, D., "Writing Secure Code, 2nd Edition", ISBN 0-7356-1722-8, Microsoft Press, 2003

[2] http://www.google.com/search?q=secure+operating+system

[3] http://video.google.com/videoplay?docid=7833143728685685343

ONLamp.com, “An Overview of OpenBSD Security”, [Internet] http://www.onlamp.com/pub/a/bsd/2000/08/08/OpenBSD.html(Accessed May 17th, 2007)

OpenBSD, “OpenBSD Security”, [Internet] http://www.openbsd.org/security.html (Accessed May 17th, 2007)

The Old New Thing

Adrian Kosmaczewski

2007-08-31

Je viens de finir de lire The Old New Thing[1]. L'auteur, Raymond Chen, a bosse dans l'equipe de developpement de Windows depuis 1995 (au moins) et il raconte les raisons de certaines decisions prises pendant le design de differentes versions de Windows, depuis 1985 jusqu'a Vista. Le livre est une compil' des meilleurs articles de son blog[2].

Et franchement, c'est a ne pas y croire.

Windows Vista possede encore des APIs pour faire tourner des applications DOS 1.0, juste pour le plaisir de la compatibilite descendante ad infinitum. Les noms des methodes de Win32 sont absolument cryptiques, impossibles a retenir, mais le gars justifie toutes et chacune de ces aberrations par des raisons historiques diverses. La registry contient des infos pour changer le fonctionnement du memory manager juste pour que Lotus 1-2-3 version 2 pour Windows (1990) tourne sans probleme sous Vista.

Je me demande comment M$ a permis que ce livre soit publie! Il fait plutot envie de ne plus jamais, jamais, developper pour Windows, dans aucun langage de programmation qui soit. Je recommande vivement sa lecture, surtout si vous avez des connaissances techniques du kernel Linux! Les descriptions du fonctionnement interne de Windows sont impressionantes, avec un niveau de detail unique.

[1] http://www.amazon.com/Old-New-Thing-Development-Throughout/dp/0321440307

[2] http://blogs.msdn.com/oldnewthing/

akosma

Adrian Kosmaczewski

2007-10-17

http://www.law.umkc.edu/faculty/projects/ftrials/socrates/ifstoneinterview.html

> Homer calls them akosma. This is the negative of kosmos, whence our words “cosmetics” and “cosmos” derive. The word implies disorder and lack of grace.

I haven’t blogged for a while. I’ll be back soon with great news. Stay tuned!

Riding the Rails Again

Adrian Kosmaczewski

2007-10-18

It feels soooooo good to :)

Let me introduce you to Parking Friend[1]. This website, which I had the pleasure to design and develop, belongs to some friends of mine, currently starting their own valet parking service in Geneva. Located not far from the airport, Jake, Dieter and their team will take care of your car for a small daily fee, for as many days as needed, meeting you at the airport (or anywhere else, for that matter) when you leave and when you return, cleaning up your car and even doing some shopping for you if you need. Handy, easy, relaxing.

Technically speaking, this is my first public, mainstream Ruby on Rails application. What can I say? It has been a delight to create by all means. It took me two weeks to do it, working… on "rails" precisely :) in the morning and evening trains while going to and from work, as well as during the weekends.

Of course the whole thing was typed on TextMate[2], stored in a Subversion[3] repository, and deployed via Capistrano[4]. The choice of technologies could not be better: TextMate is the most incredible IDE I've ever used, Subversion is always responsive and reliable (of course I was the only one using it but anyway!) and Capistrano… my goodness. I wonder how did I do before to deploy applications: **"cap deploy"** and you're done.

The logo and marketing banner were created on Inkscape[5] as an SVG file, and then retouched on Gimpshop[6] into transparent background PNG files. Everything standard, cross-platform and open-source; I don't want to lock people into proprietary stuff right now at the beginning.

The coordination with Jake and Dieter was done through Basecamp[7]

[1] http://parking-friend.com/
[2] http://macromates.com/
[3] http://subversion.tigris.org/
[4] http://capify.org/
[5] http://www.inkscape.org/
[6] http://plasticbugs.com/?page_id=294
[7] http://basecamphq.com/

for the project management stuff and Tiki[8] for bug tracking. Again, I can't stress this too much: Basecamp is a joy. It is easy. It is fast. It is great. You cannot do better for these kind of projects. It's simply perfect. I've bought a paying subscription to Basecamp after this live test, and I highly recommend you to take one too for managing these kind of projects.

I have tested the application under the following browsers and operating system combinations (yes Patrick! It works on Konqueror too!!):

- Mac OS X:
  - Safari 3.0.3
  - Firefox 2.0.0.7
  - Opera 9.24
- Windows XP Service Pack 2
  - Internet Explorer 6
  - Firefox 2.0.0.7
  - Opera 9.24
- Kubuntu Linux 6.10
  - Konqueror 3.5.6
  - Opera 9.24

I will add IE7 and the next version of Ubuntu as soon as I have them ;)

By the way, with Parking Friend I learnt that one of the best ways to achieve cross-browser compatibility is to start writing meaningful markup from the start. **Yes, semantically meaningful.** And this is a lesson I got after reading this incredible book called Transcending CSS[9]. I refreshed 10 years of HTML and CSS knowledge thanks to it, and boy I am happy I did it.

Of course, I haven't done all the work by myself; I used the following plugins in this application:

- Painless PNG Rails plugin[10]
- Streamlined[11]
- Act as Authenticated[12]
- Calendar Date Select[13]
- AssetPackager - JavaScript and CSS Asset Compression[14]
- Simple Captcha 1.0[15]
- rolerequirement[16]

[8] http://tikiden.com/

[9] http://www.transcendingcss.com/

[10] http://wheremydogs.at/articles/2007/04/30/the-rails-way-painless-png-in-ruby-on-rails

[11] http://streamlinedframework.org/

[12] http://technoweenie.stikipad.com/plugins/show/Acts+as+Authenticated

[13] http://code.google.com/p/calendardateselect/

[14] http://synthesis.sbecker.net/pages/asset_packager

[15] http://expressica.com/2007/03/23/simple_captcha_1_0

[16] http://code.google.com/p/rolerequirement/

- Exception Notifier[17]
- FlexTimes[18]
- Calendar Helper[19]

The most amazing of these plugins is, without any doubt, AssetPackager[20] (well, they are all incredible in their own, but this one really stands out). It allows you to compress JavaScript and CSS files, which reduces the number of HTTP requests, the size of the page and accelerates the speed of rendering. Used together[21] with YSlow[22] and the Yahoo! Performance Guidelines[23]... well, your app rocks, and users are happy with it. I could reduce the total download size of the page by 60% after installing it...

Another nice one is to have the Exception Notifier[24] plugin, informing me automatically about any error in the application; whatever happens, I receive an error message in my inbox, telling me what happened, when, how and even why! This is step 1 of Joel's Hard-assed Bug Fixin'[25] strategy, which is the path to remarkable customer service[26].

Finally, with Streamlined[27] I could add a complete administration backend to the system in just one hour. Now my customers can change lots of things in the system without me having to worry about it.

I have also used the following RubyGems:

- RedCloth[28] by whytheluckystiff
- RMagick[29]

And some JavaScript goodies too:

- Prototype[30]
- Script.acoul.us[31]
- Accordion v2.0[32]

Finally, the whole application has been "frozen" through rake

[17] http://dev.rubyonrails.org/browser/plugins/exception_notification/README
[18] http://rubyforge.org/projects/flextimes/
[19] http://wiki.rubyonrails.org/rails/pages/Calendar+Helper+Plugin
[20] http://synthesis.sbecker.net/pages/asset_packager
[21] http://www.slashdotdash.net/articles/2007/07/31/rails-performance-tip-using-yslow
[22] https://addons.mozilla.org/en-US/firefox/addon/5369
[23] http://developer.yahoo.com/performance/index.html#rules
[24] http://dev.rubyonrails.org/browser/plugins/exception_notification/README
[25] http://www.joelonsoftware.com/articles/fog0000000014.html
[26] http://www.joelonsoftware.com/articles/customerservice.html
[27] http://streamlinedframework.org/
[28] http://whytheluckystiff.net/ruby/redcloth/
[29] http://rmagick.rubyforge.org/
[30] http://www.prototypejs.org/
[31] http://script.aculo.us/
[32] http://stickmanlabs.com/accordion/

`rails:freeze:gems`. Of course! This way, I do not have to worry about upgrading RubyGems in the server in the future, which would break the application.

To summarize, working with Ruby in general is a joy. The language is intuitive, pretty fast, has a great library, and Rails throws a lot more of extremely usable stuff into it. I had a level of productivity that I've never seen before; and the whole thing is not even 1000 lines of code long (tests included). I can export the database to other servers if I wanted (thanks to migrations[33]) and the code size makes the whole thing extremely maintainable for the future. I can test the whole system, using unit, functional and integration tests all at once.

A dream platform, really. And open source, free, etc, etc, etc. You know the song ;)

Update, 2008-10-20: I've just checked that the site also works with the version of Opera for the Wii[34] and on my wife's Palm handheld. Standards rock!

[33] http://wiki.rubyonrails.org/rails/pages/ActiveRecordMigration
[34] http://www.opera.com/products/devices/nintendo/

Rethinking the Corporate World

Adrian Kosmaczewski

2007-10-31

In Buenos Aires I've studied corporate management (I did, shame on me), and as part of that, I had to learn about all the different identified types of organizations: matrix-based, pyramidal, military, organic, etc. Afterwards, books like Peopleware[1] made me rethink these concepts, particularly when seeing the pityful state of some software development companies here and there.

I mean, except some unusual exceptions, our work environments typically suck. Deeply. Nobody gives a damn about your ideas, and you'll just have to sit there in crowdy and noisy open spaces, and do the stupid things that you're told to do, and everything is a horrible command and conquer[2] experience. Working in this free market, post-Berlin wall world, the only choice you're left with is **suck it up or leave.**

Welcome to the free world. You're free to starve or to choose who to submit your soul to during 40 hours per week[3].

In the software development field, changing jobs is a comparatively easy thing to do, with the few exceptions of the software crisis in the mid 70s, at the end of the 80s and between 2002 and 2004, but in any case those crisis happened during short periods of time. But in other industries, people, for many reasons (mortgage, family, etc) they have to stick with horrible workplaces, awful jobs, incredible amounts of stress and awesome levels of burnout. Why does it have to be this way?

And you know what? It doesn't have to. Some company out there, in a more "brick and mortar" industry than software, thinks that the current way of doing things is wrong. If you haven't heard about Ricardo Semler[4], go and watch this movie from the MIT[5]. **It is amazing.** This guy, CEO of a industrial company in Brazil for the last 25 years, has totally changed the way things are done, and brought some extremely

[1] http://www.amazon.com/Peopleware-Productive-Projects-Tom-DeMarco/dp/0932633439

[2] http://www.joelonsoftware.com/articles/fog0000000072.html

[3] /blog/killing-in-the-name-by-rage-against-the-machine/

[4] http://en.wikipedia.org/wiki/Ricardo_Semler

[5] http://mitworld.mit.edu/video/308/

innovative ideas to his company, which has since sustained a 900% growth (!) with as little as 2% turnover (!!).

Now go watch it (it's only 40 minutes long), and I hope, learn something. I think that the Swiss business environment really needs to change, and Semler's approach might work. And by the way, the guy is really funny!

(Through 37signals' blog)[6]

[6] http://www.37signals.com/svn/posts/649-inspiring-ricardo-semler-lecture-at-mit

A Simple Recipe for Podcast Success

Adrian Kosmaczewski

2007-11-10

I am subscribed to quite a few podcasts and screencasts here and there. And I've come up with a very basic (albeit limited and you could even say irrational) way of determining which to keep listening and which to throw away immediately:

The quality of the material… and the voice of the speaker.

I'm not Pavarotti nor Alfredo Caruso, but some voices just irritate me. I just experienced this through the Heroku screencasts[1]; the guy's voice is not really nice (at all), kind of creepy even, hard to follow, I don't know how to describe it. It is annoying to follow a 10-minute presentation like this; really, I'm sorry, but that's how I felt it, even if his service seems really interesting and I might even try it in the future.

Compare now with Ryan Bates of Railscasts[2]: his voice is adapted, serious yet young, with the right pitch and speed. It makes following the explanations easy, moreover taking into account that I'm not a native English speaker. The Railscasts are a perfect example of what I like in podcasts and screencasts: short descriptions (15 min max) of extremely useful features, with practical uses and with some background as well to get the idea. The site (and Ryan) is absolutely brilliant.

As I said, is a purely subjective point of view, but that's (one) of the criteria I use to decide whether to keep listening to a podcast / screencast or not. The other being the contents, of course; throw in a nice voice spitting nonsense and you won't have much better luck than the Heroku guy.

The notable exception to this rule must be obviously David Heinemeier Hansson[3]; his first videos showing how to do a weblog in Rails in 15 minutes[4] are just insane; the guy's voice is really awful, too highly pitched and somehow disturbing. But the stuff he showed was great, and I stuck with that instead :)

[1] http://heroku.com/features
[2] http://railscasts.com/
[3] http://www.loudthinking.com/
[4] http://www.rubyonrails.org/screencasts

Deliver. Now.

Adrian Kosmaczewski

2007-11-11

Every time I talk with people about Ruby on Rails[1] in Switzerland, I almost always get the same comments, no matter what is the background of the person I'm talking to:

> Yes but… what about [scalability / performance]? [I'm sure / I've read / I think / I believe / I have dreamt] that Ruby on Rails is not as [fast / scalable / powerful] than [J2EE / .NET / PHP / ASP / CGI / WebObjects / Python / Perl]

It's very funny indeed, for many reasons:

1. None of these comments came from people running something like Facebook, or at least any other site with more than 10000 visits per month;
2. None of the people who said something like the above has tried Ruby on Rails, beyond the 15 minute blog thing, which everyone seems to have done.

It seems to me that there's a problem here.

Twitter[2] runs on Rails. And that's a heckuva lot of people posting updates every second. OK, this hasn't been a smooth ride[3] at all, but Rails brings you productivity and maintainability, and this, more often than not, outweighs performance. The creators of Twitter did not know that they would be so wildly[4] successful[5], in a way that has somehow redefined it as a new category of asynchronous communication. However, I think that if they had paid more attention about scalability from day zero, you know what? Maybe they wouldn't have **delivered Twitter at all**.

And you get customers, visibility and market position thanks to delivering products, not just by having a nice architecture.

And Twitter is not the only one: ManiaTV[6] has more than **10 million**

[1] http://rubyonrails.org/

[2] http://twitter.com/

[3] http://www.radicalbehavior.com/5-question-interview-with-twitter-developer-alex-payne/

[4] http://twitterposter.com/

[5] http://twittervision.com/

[6] http://maniatv.com/

visits a month (source: HappyCodr[7]). And Basecamp[8] is used by thousands of users every day. Like Odeo[9], Shopify[10], Fluxiom[11] or Strongspace[12]. **Rails is ready for the real world. And it kicks your platform's a** every day.**

I know of a couple of rewrites in this precise moment: somewhere, out there in the wild, there is some team rewriting a PHP application in J2EE just because it doesn't scale. Another rewriting a successful ASP application done in VBScript into ASP.NET and C# just because the code is a mess and CIO Magazine[13] says that .NET is better. Another trying to rewrite a C++ application in a J2EE just because that's how it should be done.

In my experience, rewriting is often just wishful thinking. None of these teams are ready to finish, none of these teams are ready to show anything to a potential customer. **They cannot deliver.** Deliver first, worry about performance later. I wish you luck and users and subscriptions and lots of worries about performance with all my heart!

Don't get me wrong: I believe and always have believed that architecture is an important asset in the development process[14]. However, when it gets in your way to deliver a product to your customer, it becomes a **liability**.

> Release early in the project and release often. Just like your application, your deployment doesn't need to be perfect from day one. You can start simple and grow into more sophisticated deployment strategies over time.

James Duncan Davidson[15], read in Peter Marklund's Rails 101[16].

By the way, Parking Friend[17] in on HappyCodr[18] too ;) The application is servicing around 100 customers per day after three weeks online (according to Google Analytics[19])... and for the moment it's doing well! I've Globalized[20] it recently, and also added credit card support with the ActiveMerchant[21] plugin. All under 150 hours of work (around 19 days of full-time work).

---

[7] http://www.happycodr.com/folio/show/8567
[8] http://www.basecamphq.com/
[9] http://odeo.com/
[10] http://www.shopify.com/
[11] http://www.fluxiom.com/
[12] http://www.strongspace.com/
[13] http://www.cio.com/
[14] /blog/about-software-architectures-and-the-ieee-1471-standard/
[15] http://duncandavidson.com/
[16] http://marklunds.com/articles/one/374
[17] http://parking-friend.com/
[18] http://www.happycodr.com/folio/show/8476
[19] http://google-analytics.com/
[20] http://www.globalize-rails.org/globalize/
[21] http://www.activemerchant.org/

Commentaire Sur Profession-Web

Adrian Kosmaczewski

2007-11-13

J'ai posté le commentaire suivant dans un article paru aujourd'hui sur Profession-Web:

> "On n'est là pour lui, car C'EST LUI QUI ENGAGE."

Je pense que l'état d'esprit de la phrase ci-dessus explique le turnover élevé et le bas rendement en général de l'industrie du software et du web en Suisse (particulièrement dans le consulting). Je ne suis en aucun cas d'accord avec cette façon de voir les choses.

L'employeur me paie un salaire, certes, mais il faut être clair sur un point: en tant qu'employé je lui offre 40 heures par semaine de mon temps CPU. Et ce n'est pas peu, compte tenu des open-space bruyants, des interruptions sans cesse et des horribles environnements de travail que certaines entreprises offrent a ses employés. Sans compter celles qui se donnent un plaisir de payer des salaires bien en-dessous de ce que le marché offre, ou qui n'offrent aucun programme de développement de carrière, sous la justification de voir les développeurs comme des machines à taper du code et rien d'autre.

On ne travaille pas POUR un employeur, mais AVEC un employeur. La différence est ENORME. Ce n'est pas une relation unidirectionnelle, mais bien bidirectionnelle.

Lors d'un entretien, c'est aussi au "candidat" de voir en quelques minutes si le cadre de travail, le projet et surtout les gens, sont agréables, intéressants, pour savoir s'il vaut la peine de signer le contrat. L'employeur doit donc aussi préparer l'entretien, chaque jour, en faisant de son entreprise celle qui sortira du lot, pour que les meilleurs viennent à lui.

Le livre "Peopleware" a été écrit il y a 20 ans exactement, et pourtant peu de professionnels RH en Suisse romande en ont entendu parler (encore moins lu). Le marché du soft et du web n'est pas similaire aux autres: le turnover peut tuer une entreprise qui compte sur le génie de ses employés, mais il faut savoir aussi que c'est la branche de l'industrie dans laquelle il est le plus simple du monde de changer de travail.

Alors, pas d'autre solution: il est temps de changer les mentalités.

Tant que les sociétés ne se rendent compte (à la façon de Apple, Sun, Microsoft ou Google) qu'il faut créer un cadre de travail exceptionnel pour y accueillir des gens exceptionnels, tout cela pour créer des services et des produits exceptionnels, qui vont générer des bénéfices exceptionnels, les entretiens d'embauche suivront un cours plutôt "classique", avec une claire distinction "employeur / employé" basés sur une hiérarchie digne du moyen âge, et non pas du 21ème siècle.

Git

Adrian Kosmaczewski

2007-11-29

If you haven't tried git[1], you should. Git is a "distributed version control" system, that is, similar to Subversion with the big difference that... you do not need a server. There are only clients, any of them, and you can pull and push changes to and from other repositories from your project colleagues. The git Wikipedia entry[2] does a much better job than me to introduce the subject :)

In Mac OS X, I've just downloaded the source code tarball[3] with the latest snapshot of the code, and it compiled out of the box. Just the classical operations:

```
$ ./configure
$ make
$ sudo make install
```

Then I went through the tutorial[4], and frankly, I loved it. The Everyday GIT Guide[5] is excellent too to understand the idea of this beautiful piece of code.

It's too cool, really. Lightweight, and damn fast. No wonder why everyone's talking[6] about it[7] lately[8]!

[1] http://git.or.cz/
[2] http://en.wikipedia.org/wiki/Git_(software)
[3] http://kernel.org/pub/software/scm/git/git-1.5.3.6.tar.gz
[4] http://www.kernel.org/pub/software/scm/git/docs/tutorial.html
[5] http://www.kernel.org/pub/software/scm/git/docs/everyday.html
[6] http://speirs.org/2007/07/19/a-subversion-user-looks-at-git/
[7] http://www.sanityinc.com/articles/rails-on-git
[8] http://www.evenflow.nl/2007/10/15/webistrano-git-and-capistrano-210/

Miopía

Adrian Kosmaczewski

2007-12-06

Reacción a algunos comentarios debajo de la nota “Gran Bretaña endurece las condiciones de los inmigrantes” en Clarin:

“Deberiamos hacer lo mismo”... “Me parece estupendo”... “Perfecto”... parece mentira lo que se lee en los comentarios. Puede ser que nos estemos encerrando todos? Cada uno en su rincon de mundo, teniendole miedo al otro, haciendole la vida imposible al viajar, al moverse, al tratar de zafar de la miseria por cualquier medio?

No es una cuestion de “seguridad”, no me vengan con eso; es una cuestion de que nos tenemos miedo unos a los otros, y no nos queremos hacer cargo de una verdad muy simple: el bienestar de la pequeñisima fraccion de la humanidad que vive “bien” DEPENDE EXCLUSIVAMENTE de que otros miles de millones no tengan nada de que comer. NADA. Y para no tener que hacerse cargo de esto, nada mejor que una buena aduana y fronteras con alambre de pua y perros policia y detectores de metales y mil cosas por el estilo.

La raza humana debe recordar que estamos de paso en esta Tierra, y que encerrar un par de personas detras de una frontera no ayuda en nada al bienestar del otro. No es mi trabajo de hoy que esta en juego con estas politicas: es el mundo que le dejaremos a nuestros niños. Es increible que seamos tan miopes y ridiculamente miedosos del otro.

Y al que comento que cuando se inunde Gran Bretaña por culpa del recalentamiento global, me gustaria saber que piensa al saber que Buenos Aires tambien quedara abajo del agua (ya se inunda con dos gotas de lluvia, imaginate con un par de icebergs menos, sonaste). Me gustaria ver la cara de los cordobeses, riojanos, mendocinos y patagonicos, cuando millones de porteños tengan que migrar hacia el interior del pais.

Las migraciones son parte de nuestra naturaleza. Evitarlas es algo que ya se intento, que ya no funciono, y que tampoco funcionara esta vez; aun peor, nos llevara a una destruccion total.

(este comentario lo habria puesto entero en la pagina, pero el maximo permitido es de 240 caracteres, asi que por eso lo puse completo aqui)

About Operating Systems, Abstractions and APIs

Adrian Kosmaczewski

2007-12-15

Charles Petzold, in its book “Code”, states the following:

> In theory, application programs are supposed to access the hardware of the computer only through the interfaces provided by the operating system. But many application programmers who dealt with small computer operating systems of the 1970s and early 1980s often bypassed the operating system, particularly in dealing with the video display. Programs that directly wrote bytes into video display memory ran faster than programs that didn’t. Indeed, for some applications - such as those that needed to display graphics on the video display - the operating system was totally inadequate. What many programmers liked most about MS-DOS was that it ‘stayed out of the way’ and let programmers write programs as fast as the hardware allowed.

(Charles Petzold, “Code”, pages 332 & 333)

This paragraph shows the state of things during the MS-DOS & early Windows versions timeframe (from late 1970s until 2000 approximately). During this time, programmers could directly access computer memory, bypassing the APIs offered by the operating system, and thus having total control of the hardware.

This shows two different trends in computer programming, one that respects the functionality offered by the operating system, and another that bypasses it. There are advantages and disadvantages to each approach, and the following paragraphs shows some of them.

The Conflict

The Apple Macintosh (1984) and the NeXT computer (1989) were among the first systems to introduce a complete API (Application Programming Interface[1]) that completely shielded application developers from directly accessing the hardware on which the application ultimately runs. In the case of the Apple Macintosh, this API could be

[1] http://en.wikipedia.org/wiki/API

programmed in Pascal, while for the NeXT it was using the Objective-C language.

The tradeoff between the MS-DOS approach and the API one can be resumed in three areas: **performance, portability & maintenance, and security.** The first one, considered critical in the 80s, has been one of the major factors of the success of the "compatible IBM PC" + MS-DOS platform; similar applications could ran faster than in Macintosh environments; also, a bigger number of games (which heavily use graphics) was available for that platform, and this also led to a majority of people to choose it.

Direct access to the hardware brought a first problem, of maintenance and portability; indeed, software written this way was too hard to port to different operating systems or processor architectures, since it relied heavily in the availability of certain hardware interruptions and circuitry. While, on the other hand, Apple Macintosh software created for the first version of the Mac OS (1984) could run seamlessly, without recompilation (just copy the executable and double-click on it), until Mac OS 9 (1999).

Microsoft Windows

Regarding Microsoft Windows, the situation is slightly more complicated. Windows began its life as a GUI around MS-DOS in 1985, and from its version 95 it gradually became a more independent system, but never truly becoming a multi-threaded, multi-tasking operating system. This system used to support old MS-DOS software, allowing it to run natively:

> I first heard about this from one of the developers of the hit game SimCity, who told me that there was a critical bug in his application: it used memory right after freeing it, a major no-no that happened to work OK on DOS but would not work under Windows where memory that is freed is likely to be snatched up by another running application right away. (…) They reported this to the Windows developers, who disassembled SimCity, stepped through it in a debugger, found the bug, and added special code that checked if SimCity was running, and if it did, ran the memory allocator in a special mode in which you could still use memory after freeing it.
>
> This was not an unusual case. The Windows testing team is huge and one of their most important responsibilities is guaranteeing that everyone can safely upgrade their operating system, no matter what applications they have installed, and those applications will continue to run, even if those applications do bad things or use undocumented functions or rely on buggy behavior that happens to be buggy in Windows n but is no longer buggy in Windows n+1. In fact

> if you poke around in the AppCompatibility section of your registry you'll see a whole list of applications that Windows treats specially, emulating various old bugs and quirky behaviors so they'll continue to work.

(Joel Spolsky, 2004)

As you can see, direct hardware access is not only a problem for applications developers... it was one for Microsoft as well.

Simultaneously, Windows NT[2] was started in 1988 by another team, largely composed of engineers that worked in the Digital Equipment Corporation OpenVMS system. The NT kernel features, among several distinctive characteristics, the HAL (Hardware Abstraction Layer):

> A hardware abstraction layer (HAL) is an abstraction layer between the physical hardware of a computer and the software that runs on that computer. Its function is to hide differences in hardware and therefore provide a consistent platform to run applications on.

(Wikipedia, 2006)

The NT's HAL abstracts the complete hardware beneath, and the only way to access hardware functionality is through the API. The NT team approach was radically different to that of the "classic" Windows team; they would not fix compatibility issues application per application, but would rather define an API upfront and publish it to the developers.

The NT kernel has since replaced the older 9x kernel, from version XP onwards. This shift broke many old software packages, that are not able to run properly (if at all) in the new versions of Windows. Only those programs that use the Windows API[3] exclusively are able to run properly (I have a copy of Lotus Improv 3.1[4], bought in 1992, that I used to run under Windows 3.1, and that runs perfectly well under Windows XP...!)

Present Situation

Nowadays the "performance" characteristic named above has been replaced by the security concern; code that can access directly the computer hardware without permission checks (particularly in times of the Internet) can be potentially extremely dangerous: chapter 4 of the book "Hacking Exposed" (ISBN 0-0721-2127-0), by Joel Scambray, Stuart McClure and George Kurtz exposes tens of different vulnerabilities caused by direct hardware access known to Windows 95, 98 and ME - that is, the non-NT kernel. In other terms, such systems connected directly to the Internet are simply too vulnerable to be safely

[2] http://en.wikipedia.org/wiki/Windows_NT
[3] http://en.wikipedia.org/wiki/Windows_API
[4] http://en.wikipedia.org/wiki/Lotus_Improv

usable.

The conflict between calling an API or accessing the hardware directly today has been won by the API approach; consumer operating systems, at least, have taken this road and there is no turning back. The advantages are evident; the same source code base can be used to build the same application in several different platforms, lowering maintenance and support costs; platform vendors can improve performance and security reducing the impact in existing software packages; application developers can share knowledge, tips and tricks; security is built from the ground up.

On the other side, the growth in capacity of operating systems make limited resources to appear as unlimited: memory (using paging on disk), printers (using print queues) or even screen desktop space (using multiple desktops such as the GNOME[5] approach, or on-screen gadgets such as Expose[6] on the Mac).

Conclusion

Currently there are APIs for graphic manipulation, such as OpenGL[7], that allow software such as Google Earth[8] to run in different hardware architectures using the same code base. This is possible thanks to the higher power of today's hardware, and to the advances in operating system design, that make the overhead of API calls a non-significant portion of the overall CPU time needed to execute programs. Thus, the conflict has largely been resolved, in my opinion.

References

Charles Petzold, "Code - The Hidden Language of Computer Hardware and Software", 2000, Microsoft Press, ISBN 0-7356-1131-9 (website[9])

Joel Spolsky, "How Microsoft Lost the API War", June 13, 2004 [Internet], http://www.joelonsoftware.com/articles/APIWar.html, (Accessed December 14th, 2007)

Wikipedia, "Hardware abstraction layer" [Internet], http://en.wikipedia.org/wiki/Hardware_abstraction_layer (Accessed December 14th, 2007)

[5] http://www.gnome.org/
[6] http://www.apple.com/macosx/
[7] http://www.opengl.org/
[8] http://earth.google.com/
[9] http://www.charlespetzold.com/code/

Factors of Software Project Quality

Adrian Kosmaczewski

2007-12-16

I strongly consider that the following three items are of high relevance for software project quality:

- Developer workplace conditions
- Tracking data of past projects
- Management commitment to quality

In this article I will give an overview of them, providing some personal experience about each.

Developer Workplace

Enhancing the workplace of software developers, is an often overlooked fact that has to do with both human resource management and quality management; both aspects are affected when working conditions are not properly met.

The classic book “Peopleware” focuses on the problem of the inadequate developer workplace. DeMarco and Lister correctly point out the highly intellectual nature of software development, and the fact that the most successful software companies are those that provide developers with quiet workplaces.

“Open spaces”, in this sense, are one of the most important reasons to blame for low output quality, and if I have to refer to my own personal experience, I can only confirm this fact. The places where I have been able to deliver my highest quality work are those where noise, interruptions and distractions were reduced to the minimum. This allows developers to reach a state of “flow”, where the notion of time seems to disappear completely.

In this state, the level of concentration is maximum and the quality of the work is usually the highest; software developers are able to handle in their heads lots of elements, that have not only to do with the lines of code that they are currently editing, but also with their relationships and dependencies. This notion of “dependency” is key to software architecture and development, and the best software developers are those that can hold and remember the relationships between the current lines under their eyes and those in other modules,

so that they effectively reduce coupling and the risk of bugs, while enhancing encapsulation and reuse, all factors that have to do directly with the quality of the code produced.

Moreover, good working conditions reduce turnover, which is one of the most extreme, expensive and feared forms of corporate memory loss.

Past Project Data Tracking

Smart companies have good memory. They are able to tell, from past projects, the rate of success and failure, the context, the errors made, the level of accuracy of their methods, and the success rating of the most accurate analysts. All of this, when systematically stored and retrieved, helps organizations to have a higher level of quality, since from the very beginning they can tell what works and what does not work. "Been there, done that"; not all corporations are able to say this phrase.

I have worked in companies of both kinds, and I can say that the level of "corporate awareness" helps a lot to define project plans and to make good estimations for projects. For every project, a database of similar projects, coupled with experienced teams helps making good decisions in future endeavors. These good decision have a direct impact on the perceived quality of these organizations.

Management Commitment

One of my past employers was certified ISO 9000, while another was going through the evaluation for some CMMI certification. And I can tell that certificates are worth nothing if the management does not have a strong commitment to quality.

In one case, the ISO certification was only a way to attract corporate customers, and nothing else. There was a bunch of quality procedures stored somewhere in a closed intranet (to which strangely enough, developers had no access). There was a "Quality Manager" that knew next to nothing about the day to day job of consultants. There was a complete lack of professionalism from the project managers, who relied upon the ignorance of some big clients to get contracts and some cash. That was it. Oh, and we were told, once a year, to be kind to the ISO people that came to verify that the quality procedures were still in place, in order to renew the certification (this is absolutely true). All of this impacted our day-to-day work enormously, needless to say, since the image we got from our managers was that quality was only good for brochures; interacting with them was far from pleasant, and the output of our work as teams was less than desirable.

On the other company, they had set up every possible practice in the road to certification; just to give an idea, their product consisted of a couple of million lines of source code, which were built every night,

with thousands of unit and functional tests run automatically after the build; reports and API documentation were created automatically and dispatched to key stakeholders; daily stand-up meetings were held every day; internal training done every week; and the first person to follow all the guidelines and principles was the CEO, who happened to be also the architect and the biggest committer in the source control system. New features were evaluated by senior members, and change management procedures were religiously followed. The commitment to quality goes up and down the hierarchy levels as I had never before seen it, but with a strong example set by the upper levels. The quality of the source code is unparalleled (the product is 10 years old already), and I think that those who worked with complex system know how important is, for maintenance, performance and readability purposes, to have strong quality standards with it.

As in other aspects of project management, the commitment of upper levels of the hierarchy is fundamental for the quality of the final output as well.

Conclusion

Having better working conditions, with a management aware of past project experiences, and commited to quality enhancements are important factors, that can impact positively the final quality of a project.

References

Tom DeMarco & Timothy Lister, “Peopleware - Productive Projects and Teams, 2nd Edition”, 1999, Dorset House Publishing, ISBN 0-932633-43-9

Total Quality Management and Software

Adrian Kosmaczewski

2007-12-18

Total Quality Management is one of the founding pillars of modern mass-production economy, of which the software industry is by far the youngest (and most rebel) child. This article will provide a short discussion on some TQM principles and about their applicability to software projects.

TQM Principles

Joel Spolsky has written a brilliant introduction to software testing in his famous "Joel on Software" blog:

> Software has bugs. CPUs are outrageously finicky. They absolutely refuse to deal with things that they weren't taught to deal with explicitly, and they tend to refuse in the most childish of ways. When my laptop is away from home, it tends to crash a lot because it can't find the network printer it's used to finding. What a baby. It probably comes down to a single line of code somewhere with a teensy tiny almost insignificant bug in it.

(Spolsky, 2000)

The software industry has been a victim of its own success. Software companies have experienced the fastest growing rates in the history of capitalism, and this rush to conquer juicy worldwide markets, and to offer services to literally billions of human beings has, more often than not, been done at the expense of proper quality guidelines.

William Deming proposed 14 key principles that form the basis of what became TQM in the second half of the 20th century:

1. Create **constancy of purpose** for the improvement of product and service (…)
2. Adopt a new **philosophy of cooperation** (win-win) (…)
3. Cease dependence on mass inspection to achieve quality. Instead, **improve the process** and build quality into the product in the first place.
4. End the practice of awarding business on the basis of price tag alone. Instead, **minimize total cost** in the

long run. (…)

5. **Improve constantly**, and forever, the system of production, service, planning, of any activity. (…)
6. **Institute training** for skills.
7. Adopt and **institute leadership for the management of people**, recognizing their different abilities, capabilities, and aspiration. (…)
8. **Drive out fear** and build trust so that everyone can work more effectively.
9. **Break down barriers between departments.** Abolish competition and build a win-win system of cooperation within the organization. (…)
10. **Eliminate slogans**, exhortations, and targets asking for zero defects or new levels of productivity. (…)
11. **Eliminate numerical goals**, numerical quotas and management by objectives. (…)
12. **Remove barriers** (…) abolishing the annual rating or merit system that ranks people and creates competition and conflict.
13. Institute a vigorous program of **education and self-improvement.**
14. Put everybody in the company to work to accomplish the transformation.** The transformation is everybody's job.**

(Wikipedia, emphasis added)

In the following section, I will discuss how these principles overlap and intersect, and how they can be applied (and have been applied historically) in software companies.

Focus on the People

From the above list, some commonalities appear for all and each one of the principles: the biggest one in my opinion is the focus on the people. As such, knowing that software is a pure mind product, completely intangible and tremendously flexible, I think more than ever that software is a human & social process. This focus in people is particularly evident in principles 2, 6, 7, 8, 9, 10, 13 and 14.

For example, reducing the causes of internal and external conflict in the organization (principles 12, 9, 2), makes software developers able to collaborate with other departments or companies, creating integrated software suites that become de facto standards in their respective industries. A very extreme example of this phenomenon is Microsoft's Office System, which was created from separate products created by different teams, such as Word and Excel. The synergy created by the developers working together, not only unifying the UI look & feel or the programming model, but also increasing the interoperability of both systems, created a product that effectively is much more than just two products together.

Moreover, the Peopleware book by DeMarco and Lister showed that historically, the most successful software companies have been those that excelled in creating a human-centric environment:

> In 1982, (Mitchell Kapor) founded Lotus Development Corporation, for which he is most noted. While there, he revolutionized corporate workplace culture by making diversity and inclusivity top priorities in his goal for creating an environment that attracted and retained employees. There were many "firsts" for Lotus, including being the first company to sponsor an AIDS Walk event in the mid-80's and refusing to do business with South Africa due to Apartheid.

(Sterling-Hoffman)

Thanks to a sharp hiring process, a series of innovations in their flagship spreadsheet product, and a progressive corporate culture, Lotus dominated the software landscape of the 80s. Today, Google follows very closely Lotus' steps (Google, 2007a), and their brilliant results in the last few years seem to confirm this trend. Google applies principle 13 very strongly, allowing their employees to use 20% of their time in their own projects (Google, 2007b). This is resulting in an incredible amount of code, used internally and also released as open-source projects, such as the MacFUSE project (Google Mac Blog, 2007):

> Google is a fantastic company to work for. I could cite numerous reasons why. Take the concept of "20 percent time." Google engineers are encouraged to spend 20 percent of their time pursuing projects they're passionate about. I started one such exciting project some time back, and I'm pleased to announce that Google is releasing the fruits of this project as an open source contribution to the Macintosh community. That project is MacFUSE, a Mac OS X version of the popular FUSE (File System in User Space) mechanism, which was created for Linux and subsequently ported to FreeBSD.

Conclusion

Deming's principles are today, more than ever, extremely important in the software industry. With a very high rate of turnover and burnout, software companies are faced with the choice of stop considering their staff as a "resource" but rather as an "asset". The most successful companies in the field are not only those that are able to cut costs effectively (following the fourth principle) but also those that arrive to empower their staff, increasing creativity, well-being and innovation, which ultimately leads to market leadership and economic success.

References

DeMarco, Tom & Lister, Timothy, “Peopleware - Productive Projects and Teams, 2nd Edition”, 1999, Dorset House Publishing, ISBN 0-932633-43-9

Google, “Top 10 Reasons to Work at Google”, 2007a [Internet] http://www.google.com/jobs/reasons.html (Accessed April 5th, 2007)

Google, “What’s it like to work in Engineering, Operations, & IT?”, 2007b, [Internet] http://www.google.com/support/jobs/bin/static.py?page=about.html (Accessed April 5th, 2007)

Google Mac Blog, “Taming Mac OS X File Systems”, January 11th, 2007, [Internet] http://googlemac.blogspot.com/2007/01/taming-mac-os-x-file-systems.html (Accessed April 5th, 2007)

Lewis, W.; Veerapillai, G.; “Software Testing and Continuous Quality Improvement”, Auerbach Publications, 2005, ISBN 0-8493-2524-2

Spolsky, J.; “Top Five (Wrong) Reasons You Don’t Have Testers”, [Internet] http://www.joelonsoftware.com/articles/fog0000000067.html (Accessed April 5th, 2007)

Sterling-Hoffman, “Opening Doors To Higher Education”, [Internet] http://www.sterlinghoffman.com/newsletter/articles/article140.html (Accessed April 5th, 2007)

Wikipedia, “William Edward Deming”, [Internet] http://en.wikipedia.org/wiki/W._Edwards_Deming (Accessed April 5th, 2007)

Erlang

Adrian Kosmaczewski

2007-12-19

As I said before[1], I like to learn a new programming language every year. I also like to read at least 6 computing-related books every year[2], but the article about those 6 will come later[3].

The reason for these two rules is twofold: first, it gives me lots of material to blog about, and helps me refine my list of preferred languages[4] :) secondly, and more seriously, learning a new programming language makes you think differently about problems. It's not just something to keep my CV updated; it's to challenge what I know, how I know it, and why I know it. It is important, and it's not easy[5]; and I'm not the only one to do it[6].

From all the candidates that I had for this year[7], I chose to learn **Erlang**[8]. Here's some of my impressions.

[1] /blog/a-new-programming-language-every-year/
[2] /blog/my-bookshelf-part-i/
[3] /blog/best-books-of-2007/
[4] /blog/preferred-programming-languages/
[5] http://weblog.raganwald.com/2007/10/challenge-of-teaching-yourself.html
[6] http://next.yahoo.net/archives/4/on-learning-new-programming-languages
[7] /blog/this-years-programming-languages/
[8] http://www.erlang.org/

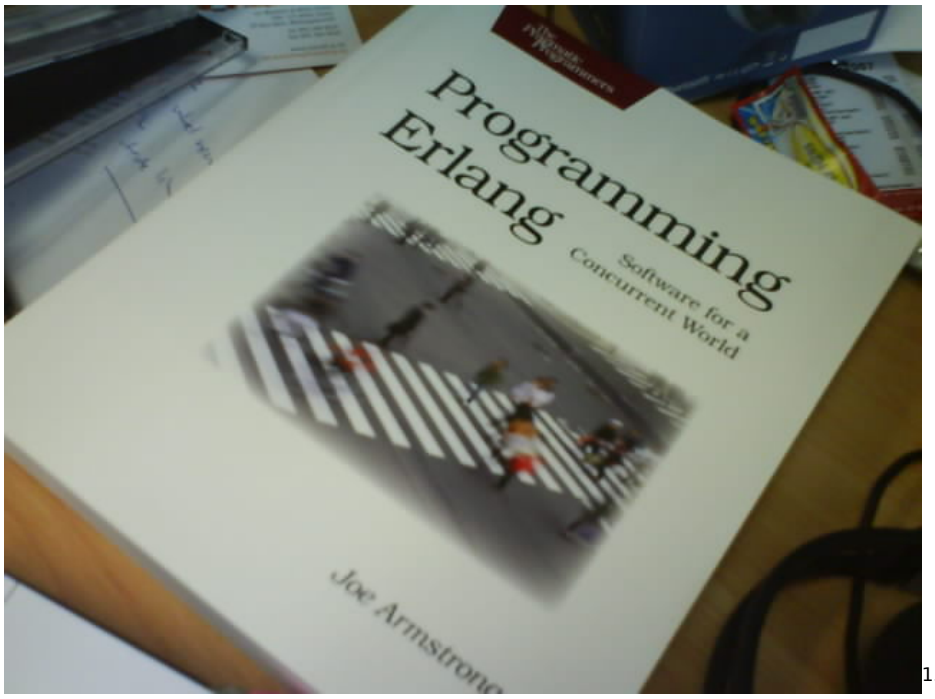
 [9]

Erlang is a functional programming language, created by Joe Armstrong (photo) at Ericsson in the 80s. The name of the language is a pun between the name of the Danish mathematician Agner Krarup Erlang[10] and the acronym "ERicsson's LANGuage". By the way, lots of things[11] are coming from Denmark lately.

The main factor that made me jump on this wagon was that the language got a lot of attention, both in 2006 and 2007. The peak of all this buzz was a couple of days ago, when Amazon unveiled a new online service called SimpleDB[12], apparently built with Erlang[13]. **The reason for this is Erlang's inherent capability to handle massively parallel programming relatively easily.** In Erlang you can create lightweight processes, each sending messages to the others, and thus divide your problems in small units that run concurrently.

Even more buzz, the release of the Pragmatic Programmer's book about Erlang[14] (written by Joe Armstrong himself) has added a strong factor to the recent popularity of the language.

[9] https://www.flickr.com/photos/mbiddulph/2037845171/
[10] http://en.wikipedia.org/wiki/Agner_Krarup_Erlang
[11] http://en.wikipedia.org/wiki/David_Heinemeier_Hansson
[12] http://aws.amazon.com/simpledb
[13] http://www.satine.org/archives/2007/12/13/amazon-simpledb/
[14] http://www.pragprog.com/titles/jaerlang



[15]

How to start with Erlang?

1. Get it working on Leopard[16] using MacPorts[17];
2. Activate the Erlang language support[18] in TextMate[19];
3. Follow a quick tutorial[20] to get the idea;
4. Check out the Yaws web server[21] build on Erlang, apparently much, much stronger than Apache under high load![22];
5. Download, install and play with ErlyWeb[23], an MVC framework similar to Rails;

More interesting reading (and viewing) about the language:

- Erlang: the Movie[24];
- What's all this fuss about Erlang?[25], by Joe Armstrong;
- Web 2.0: Shifting from "Get Fast" to "Get Massive"[26];

[15] https://www.flickr.com/photos/taniwha/1358541112/
[16] http://erlang.darwinports.com/
[17] http://www.macports.org/
[18] http://netcetera.org/cgi-bin/tmbundles.cgi?bundle=Erlang
[19] http://macromates.com/
[20] http://www.erlang.org/download/getting_started-5.4.pdf
[21] http://yaws.hyber.org/
[22] http://www.sics.se/~joe/apachevsyaws.html
[23] http://erlyweb.org/
[24] http://video.google.com/videoplay?docid=-5830318882717959520
[25] http://www.pragprog.com/articles/erlang
[26] http://www.process-one.net/en/blogs/article/web_20_shifting_from_get_fast_to_get_massive/

- Concurrency is easy[27], in Joe Armstrong's blog[28];
- More Erlang[29], by Yariv Saran[30], a guy who really knows about Erlang;
- An Introduction to Erlang[31] on the ONLamp.com O'Reilly's website.

My del.icio.us link list about Erlang[32] will keep growing... subscribe to the RSS feed to get updated of new entries on your favorite reader.

In spite of some opinions that I have seen about Erlang so far (please check a question I've asked at LinkedIn[33] about it), I have found this language **extremely interesting**; and at the moment, my list of known programming languages goes like this:

- 1992: QBasic[34]; it came bundled with my first PC!
- 1993: Turbo Pascal[35]; it was part of the official curriculum in the University of Geneva.
- 1994: C[36]; my first contact with the bracket syntax.
- 1995: Delphi[37]; I preferred more the Pascal syntax at that time... and it was an amazing environment to work with.
- 1996: Java[38]; everybody was talking about it at the time! You couldn't miss it.
- 1997: JavaScript[39]; adding interactivity to web pages was the coolest possible thing to do at the time!
- 1998: VBScript[40]; that's when I started working with MS' ASP technology;
- 1999: Transact-SQL[41]; those ASP pages were talking to a SQL Server 6.5 in the background...
- 2000: C#[42] and Prolog[43]; the first was the coolest thing since sliced bread (remember the technology previews back in August 2000?), and the second, part of the official curriculum at the Universidad de Buenos Aires...
- 2001: C++[44]; I had to learn it. It's a major step in anyone's

[27] http://armstrongonsoftware.blogspot.com/2006/08/concurrency-is-easy.html
[28] http://armstrongonsoftware.blogspot.com/
[29] http://yarivsblog.com/articles/2006/06/14/more-erlang/
[30] http://yarivsblog.com/
[31] http://www.onlamp.com/pub/a/onlamp/2007/09/13/introduction-to-erlang.html
[32] http://del.icio.us/akosma/erlang
[33] http://www.linkedin.com/answers/technology/software-development/TCH_SFT/144227-5148772
[34] http://en.wikipedia.org/wiki/QBasic_programming_language
[35] http://en.wikipedia.org/wiki/Turbo_Pascal
[36] http://en.wikipedia.org/wiki/C_programming_language
[37] http://en.wikipedia.org/wiki/Borland_Delphi
[38] http://en.wikipedia.org/wiki/Java_(programming_language)
[39] http://en.wikipedia.org/wiki/JavaScript
[40] http://en.wikipedia.org/wiki/VBScript
[41] http://en.wikipedia.org/wiki/Transact-SQL
[42] http://en.wikipedia.org/wiki/C_Sharp
[43] http://en.wikipedia.org/wiki/Prolog
[44] http://en.wikipedia.org/wiki/C++

career.

- 2002: PHP[45]; my jump to the open source world started here…
- 2003: Objective-C[46]; after I became a “switcher”, I installed the free developer tools that came with Mac OS X… and never looked back!
- 2004: Visual Basic.NET[47]; stuff that some clients required me to learn… geez.
- 2005: Ruby[48]; my flirting with Rails started that year.
- 2006: LINQ[49]; a language inside a language: or how to use C# to filter collections in-memory as if they were small databases…
- 2007: Erlang[50]; here I am now!

[45] http://en.wikipedia.org/wiki/PHP
[46] http://en.wikipedia.org/wiki/Objective-C
[47] http://en.wikipedia.org/wiki/Visual_Basic_.NET
[48] http://en.wikipedia.org/wiki/Ruby_(programming_language)
[49] http://en.wikipedia.org/wiki/Language_Integrated_Query
[50] http://en.wikipedia.org/wiki/Erlang_(programming_language)

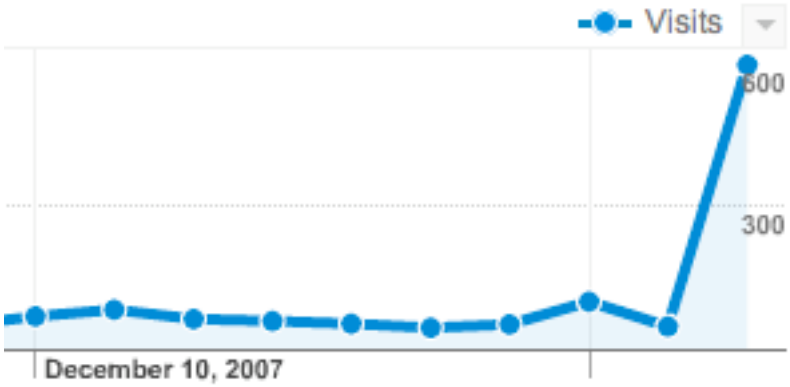
Erlang is a hot thing, indeed. Here's why.

Adrian Kosmaczewski

2007-12-20

I usually have around 50 to 80 visits per day in this blog; I'm not that famous after all, mind you; it must be the polish family name, people must have a hard time typing it ;)

Today I checked my report at Google Analytics... and this is what I saw: **more than 600 visits for yesterday alone!** Not only that, but I've had 10 comments posted in many articles of this blog, just yesterday, while usually I get one or two per week:



Apparently yesterday's Erlang article was the main drive for all of these visits:

Content Overview

Pages	Pageviews	% Pageviews
/2007/12/19/erlang/	621	15.71%
/	350	8.85%
/blogs/tech/archives/2006/02/how_to_in...	349	8.83%
/2007/11/18/installing-xubuntu-710-on-a...	178	4.50%
/2007/08/13/mercury/	118	2.98%

view report

But how did it happen? It seems that someone posted the link at programming.reddit.com and lots of people came to see:

Source/Medium	
1.	google / organic
2.	(direct) / (none)
3.	programming.reddit.com / referral
4.	google.com / referral
5.	reddit.com / referral
6.	yahoo / organic
7.	weblogs.java.net / referral
8.	profession-web.ch / referral
9.	ask / organic
10.	code.google.com / referral

It this means anything about Erlang, well, I think I must learn more about it, faster! This is amazing!

Thanks everyone who came, and I hope you enjoyed the reading. Please feel free to leave comments, in any post that you like. I love to hear your feedback on what I write.

Update, 4 hours later the publication of this article: It seems

that many of you are coming to this blog from dzone.com... welcome and thanks for your visits! I have to tell you that, at the moment, my Google Analytics stats show that more than 700 people have come here today! Thanks to all of you!!

3 years; 3 años; 3 ans

Adrian Kosmaczewski

2008-01-01

I've been blogging for three years[1] today. **Thanks to you** for your 239 comments that make this blog so unique and special; I wish you a very Happy New Year 2008!

Hoy este blog cumple tres años[2]. **Gracias a todos ustedes** por sus 239 comentarios que hacen tan único y especial a este blog; les deseo que tengan un hermoso año nuevo 2008!

Aujourd'hui ce blog fête trois ans[3]! **Merci à vous tous** pour vos 239 commentaires qui font de ce blog quelque chose de très unique et spécial; et je vous souhaite une très belle nouvelle année 2008!

:)

[1] /blog/beginning/
[2] /blog/beginning/
[3] /blog/beginning/

My First Django Project

Adrian Kosmaczewski

2008-01-11

So here it is, my first Django project: the gazillionth blog engine on the planet![1]. As if there weren't enough, right? :) Actually it was a practical and easy way to learn the Django project, and the result is pretty neat. Feel free to download[2] it, play with it, and give me your feedback. Here's a sample screenshot in Safari:

Creating this project I have had a practical experience comparing both Django and, of course, Rails. The subject is not new in this blog[3]; however, this time I could play with both frameworks and as such, I can bring my small amount of confusion in this big framework tar pit[4].

[1] blog.zip
[2] blog.zip
[3] /2007/12/05/rails-vs-django/
[4] http://www.laputan.org/mud/

Personally, I found Django and Rails **much more different than I previously thought**. Both frameworks tackle similar problems, sometimes in similar ways; however; but this does not make for a real similarity between both. They have different philosophical approaches to the problem. This must surely have to do with the underlying programming language and their philosophies.

That said, **I found Rails much easier to use.** It took me far, far less time to do a similar small application in Rails than it took me to do it on Django. And I must say that when I first used Rails, I had never seen a line of Ruby in my life. I will list some pros and cons that I found in the process:

Good things about Django:

- The "automatic administration system"; impressive! (but you can get it in Rails with some external plugins like Streamlined[5]... but in any case it's a handy thing to have it already in the system!)
- The form subsystem; great!
- Django does not force you a folder layout; nice!
- The Django Book + documentation: really great resources, nothing to say about that.
- The deployment procedure: really much easier than Rails! We're in something really close to PHP here, while Capistrano has much more to do with Java's Maven or even makefiles :)
- The native support for RSS feeds! This is a godsend.

Bad things about Django:

- No native support for "environments", like in Rails; you cannot separate easily the settings for development, testing and production (and you can add more environments if you want!)
- No native support for REST architectures, or at least "easy" AJAX + API support (like the one you find in Rails)
- The "syncdb" system that ships with Django is, at most, primitive, compared to Rails migrations.
- There doesn't seem to be a built-in infrastructure for tests.

  There is a **primitive** built-in test infrastructure. You can add "doctests" to your views and models, and this way you're "documenting and testing" at the same time. Sorry, but I don't buy this. Tests are tests, docs are docs. Rails, thanks to rakefiles and unit, functional and integration tests, seems **is** a much more advanced platform in this sense. You can get even stats of your projects with it! You can test them! Extract the docs! Everything! This is something lacking in Django. Really. Not to speak about the lack in Python of something similar to RSpec[6], which just by itself justifies the choice of Ruby and Rails in this field.

[5] http://streamlinedframework.org/
[6] http://rspec.info/

- Lack of integrated logging. Can you believe this? I had to find an external source of inspiration[7] for this.
- The naming of “Views” for “Controllers” (or said otherwise, the MTV instead of MVC thing) does not make sense to me. Even in WebObjects and Cocoa “views” are, well, “views”…
- The commands for creating, starting or using an application from the command line are slightly harder to remember (or maybe in Rails are just far too easy to remember!)
- The template system; I do not want to learn another language for that. Django uses a really limited one for this matter, and I prefer Rails’ (default) one in this case.
- Finally, Python; I just can’t get used to this language’s syntax. But I can live with that ;) It’s a matter of personal taste, simply.

Of course, these impressions have nothing to do with the power of the platform! Python is a powerful language, and many of the “newbie” observations above have to do with my own lack of knowledge about it. But first impressions count!

In any case, I liked playing with Django & Python; I’m happy to have learnt something new!

Update, 2009-02-26: I’ve posted the code in Github[8] now, and also I’ve fixed its compatibility problems, and now it works with Django 1.0.2.

[7] http://www.djangosnippets.org/snippets/16/
[8] http://github.com/akosma/django-blog-engine/

The Truth Be Told

Adrian Kosmaczewski

2008-01-22

Reg describes 99% of all available programming jobs with incredible sincerity[1]:

> You do a clerk's job, you settle for a clerk's working conditions and wages, but you take solace in the thought that you are somehow more than a clerk, because you have a university degree and the dental technician who cleans your teeth doesn't.
>
> Only everyone knows it's a sham, especially the hiring manager who puts "University degree required" in the job advertisement. He wants to hire a clerk, someone who will work long hours doing as they're told in a top-down, hierarchal command structure. Does that job sound like there is any Science involved? Of course not, everyone knows that, it's why the industry is trying to weed all of the Science out of a Computer Science degree.

I do not have a university degree. Heck, I started university 4 times and finished none (Physics, Economics, Marketing and even Computer Science! :) I have been refused jobs because of this (particularly at the beginning), but I have been given jobs because of this too (particularly lately). Having some experience in your CV pays off; the hard thing is to start without that "paper".

As a matter of fact, I'm doing an online Master's degree[2] right now, which I will finish this year, and I'm actually quite happy to have started. I've been able, in the past two years, to read books and papers that I had not heard about; I could understand some underlying issues in Software Engineering, both from technical and social points of view; I can understand more, I can learn more.

But, the truth be told, I'm also doing it to have that "paper" hanging on the wall. I know it's silly, but I want to take that step too, and that's why I've changed careers so many times. However, that is a secondary thing for me: what I am looking is something else altogether; **a bit of guidance in my own learning path**. I have followed an unusual way in my career, and looking backwards I'm happy to have

[1] http://weblog.raganwald.com/2008/01/no-disrespect.html

[2] http://www.uol.ohecampus.com/

done that (mind you, it was not at all consciously!). That's why the 6 books and the new programming language every year. It's all part of the same pattern.

Of course, this has worked out for me, and your mileage may vary. I know excellent developers both with and without degrees, and also horrible professionals with and without them. Some people need a physical teacher in front of them, while I prefer to learn alone.

A degree, like any decision that you take in life, should mean something to us, and in that sense, I find Reg's arguments enlightening.

Best Books of 2007

Adrian Kosmaczewski

2008-01-23

I have several mantras in my life. One of them is to learn a new programming language every year[1]. Another one is to read at least 6 technology-related books every year[2].

I've already talked about Erlang[3] (and boy that was the most read article ever in the whole life of this blog! More than 1600 visits just for it!) so now it's time to discuss the greatest books I've read in 2007 (ordered by preference, from more to less):

- Transcending CSS[4] by Andy Clarke[5]
- Founders at Work[6] by Jessica Livingston[7]
- The Old New Thing[8] by Raymond Chen[9]
- iWoz[10] by Steve Wozniak[11]
- Best Software Writing[12] by Joel Spolsky[13]
- Eric Sink on the Business of Software[14] by Eric Sink[15]

Transcending CSS

My big winner for this year. It's hard not to recommend this book enough, not only to those that work in the web design industry, but also to those that design, simply put. It's a beautiful book. It's a pleasure to read. It's a surprise, a bliss and a revelation, all in one.

Andy Clarke, a well-known designer in the UK, who is also now a member of the CSS committee at the W3C, tells us that HTML can

[1] /blog/a-new-programming-language-every-year/
[2] /blog/my-bookshelf-part-i/
[3] /blog/erlang/
[4] http://www.transcendingcss.com/
[5] http://www.stuffandnonsense.co.uk/
[6] http://www.foundersatwork.com/
[7] http://www.foundersatwork.com/author.html
[8] http://blogs.msdn.com/oldnewthing/
[9] http://en.wikipedia.org/wiki/Raymond_Chen
[10] http://www.iwoz.org/book
[11] http://www.iwoz.org/
[12] http://www.joelonsoftware.com/articles/BestSoftwareWriting.html
[13] http://www.joelonsoftware.com/AboutMe.html
[14] http://www.ericsink.com/bos/Business_of_Software.html
[15] http://www.ericsink.com/about_author.html

and should be semantically correct, and that you can do a lot using the standard tags already available. Stop using those <DIVs> everywhere! You can give your page a meaning to begin with, and then apply a style on top of it. You can find inspiration in everyday life, and you can make all of this a truly cross-browser experience without much effort. Forget about your HTML editor; use Notepad or TextEdit or gedit and discover a new world.

This book has a deeper meaning and raison d'être, too; **the web technologies are getting to a point of maturity as of yet unseen.** We can go beyond what we've seen so far, just sticking to standards, making meaningful designs, and caring about the user.

Absolutely enlightening.

Founders at Work

This book was one of the most hyped ones in 2007. Everyone wrote about it, starting with Paul Graham[16], Guy Kawasaki[17] or Joel Spolsky[18]. And even me[19]!

Frankly, the book is really worth every bit of the hype that surrounds it. I love computer history books (I already own a few of them[20]) and this one is, together with "Dealers of Lightning" the one I liked the most. The stories of how Lotus, Apple, VisiCalc, Firefox, PayPal or the BlackBerry appeared and grew are simply fascinating.

The book might have had more impact and hype in the entrepreneurial world, but I prefer to see it as a landmark history book (too). I've started working in the 90s, during the dot-com boom, and saw many similar patterns as those described in the book; incredible market value evaluations, products strongly marketed but born dead, and incredible stories of successes that nobody would have thought to be possible. The personal computer revolution of the 70s and the 80s has many similarities with what happened in the web revolution, and also with what happens now during the Web 2.0 hype. These are tremendous waves, that redefine the entire industry. And I think, we're doomed to relive these again and again.

The New Old Thing

I've already written about this book[21], albeit in French :) So I'll translate what I've said so far there:

[16] http://www.paulgraham.com/foundersatwork.html
[17] http://blog.guykawasaki.com/2007/03/founders_at_wor.html
[18] http://www.joelonsoftware.com/items/2007/01/30.html
[19] /blog/craving-to-read-back-to-commuting/
[20] /blog/my-bookshelf-part-iii/
[21] /blog/the-old-new-thing/

I've just finished reading The Old New Thing. The author, Raymond Chen, worked in the Windows development team since 1995 (at least) and explains the reasons behind some decisions taken during the design of different versions of Windows, since 1985 to Vista. This book is a compilation of some of the best articles in his blog[22].

And frankly, it's hard to believe.

Windows Vista still has APIs used to run DOS 1.0 applications, just for the pleasure of "ad infinitum" backwards compatibility. The names of the Win32 methods are completely cryptic, impossible to remember, but Chen justifies each and every one of these oddities by different historic reasons. The registry contains informations used to change the internal behavior of the memory manager, so that Lotus 1-2-3 version 2 for Windows (1990) could work flawlessly under Vista.

I ask myself how could M$ allow such a book to be published! It makes me wish to never, ever develop software for Windows ever again, in any programming language. I strongly recommend this book, particularly if you have technical knowledge about the Linux kernel! The descriptions of the internal workings of Windows are impressive, with a level of detail never seen before.

iWoz

It is hard to argue the fact that Steve Wozniak has invented the personal computer as we know it today. If you had any doubts (even after reading his interview in "Founders at Work"), this book will wipe them away completely.

It is written by Woz himself. Wait, did I say written? This is a told story, that almost becomes a legend at the end of the book. Woz is not modest about his feat; but he does not brag about it either. He talks about his parents, his marriages, his children, Steve Jobs, the Apple I and the Apple][, with sincerity, humor and ingenuity.

You do not need to be a fan of Apple to enjoy this book; you just need to use a computer, remember that your parents didn't, and ask yourself, how did all of this began?

Best Software Writing

This book holds the "I" numeral, but the second version has not yet been published at the time of this writing. This book is interesting in many ways; first of all, it is part of an overall tendency to write blog-based books. Joel Spolsky, Eric Sink, Raymond Chen and others are part of this trend; popular blog posts that become excellent books when put together. This book is a compilation of what Joel found the most interesting during 2004, published in agreement with the respective authors.

[22] http://blogs.msdn.com/oldnewthing/

This book is also interesting for another reason: I consider 2004 to be a pivotal year in our industry. Subversion was released that year, as were Rails and Firefox and many other popular packages. Not only that, but the whole Web 2.0 trend can be seen as a rising to the public eye in that precise moment. The book does not explicitly show these trends, but there is an overall feeling on all the best writing for that year, that something was going on. The dot-com boom was finally behind, and new things could happen again.

Eric Sink on the Business of Software

Finally, a complete hands-on resource, useful to those seeking to start a software business (yeah, like me :) The author is Eric Sink, founder of SourceGear[23], maker of Vault[24], a popular version control system for Windows marketed as a drop-in replacement for SourceSafe. He talks about all the aspects of running a software company: finance, technology choices, tradeoffs, human resources, everything.

Even if the book is primarily targeted to the US market (which makes some stuff useless in other parts of the world, particularly the legal stuff) I think it is worth a read, and again, Sink's a great writer and the book is clear and concise.

And what about 2008?

This is what I've already started to read this year:

- The Cathedral and the Bazaar[25], by Eric S. Raymond[26]
- Programming Erlang[27], by Joe Armstrong[28]
- Prototype and script.aculo.us[29], by Christophe Porteneuve[30]

Any suggestions for more books welcome! Feel free to leave me a comment below.

[23] http://www.sourcegear.com/
[24] http://www.sourcegear.com/vault/
[25] http://www.oreilly.com/catalog/cb/
[26] http://www.catb.org/~esr/
[27] http://www.pragprog.com/titles/jaerlang
[28] http://armstrongonsoftware.blogspot.com/
[29] http://www.pragprog.com/titles/cppsu
[30] http://www.tddsworld.com/blogs/eapc/

Creative Processes

Adrian Kosmaczewski

2008-01-24

It always start with a white space, like this blog post.

It could happen on a sheet of paper, or lately an empty <textarea>, but the feeling is the same. You get the first flow of ideas, rushing through your head. That's why I always have a small Moleskine notebook in my pocket; you never know when these things will happen, how long they will last, and when you'll have another one. Paper has an obvious advantage on computers: you do not need electricity.

I do not "have" an idea; the idea has me. It comes, and then it goes. My task is to write it down, or let it go. I choose.

Jadis you would have started writing in the old-fashioned way; pen, pencil, paper, but the keyboard has got our attention. It's a different feeling; before, just one hand was to held responsible for my stuff; now it's every one of my fingers. Simultaneously. It has an advantage, which is to make things go faster. I can write faster, almost as fast as I think. But I need electricity, though.

I can create not only texts but also code. I can later make that code run on the computer, on mine or even on yours. I can talk to those computers, to make them do what I want. We're new wizards, in a world that would turn crazy any 16th century man. We talk to the machine, and it answers us.

Then I go back, I erase, I reorder paragraphs, I re-read, I spellcheck, I smile, sometimes I tear off the page and start from scratch (the equivalent of selecting "Edition / Select All" & hitting the delete key).

Sometimes I publish, sometimes you read me. There is a nice randomness in all of this, and I clearly enjoy it. Thanks for being there.

That Nice Freedom of Modifying Software

Adrian Kosmaczewski

2008-01-29

One of the best learning tools I have found in my career is to take someone else's code, and to modify it slightly to see what happens, to play with it, and eventually to release that code in this blog, or send it to the original author, fixing it somehow or adding some feature:

- I fixed a small bug[1] in Matt Gemmel's iCal Birthday Shifter[2] (written in AppleScript);
- I am running a custom version[3] of Apple's own World Clock Dashboard widget that temporarily fixes the bad time zone information given the recent shift in my birth country[4] (pure JavaScript);
- I have a custom Twitterlex version[5] (again, pure JavaScript);
- And finally, this blog is using a custom version of Erik Range's SyntaxHighlighter WordPress plugin[6] (written in PHP).

This last one, is a small modification so that you can use the "firstline" CSS modifier, like this:

```
[source:javascript:firstline(150)]
// your code here...
[/source]
```

You can download this modified version of the syntax.php file here[7] or get the diff file with the modifications here[8].

This is the freedom of modifying software[9], as RMS[10] talked about. This is how the community goes forward.

---

[1] /2006/11/01/a-small-moment-of-glory/

[2] http://mattgemmell.com/2006/10/29/software-birthdays

[3] /2007/12/30/argentina-world-clock-dashboard-widget/

[4] http://www.theinquirer.net/gb/inquirer/news/2007/12/30/dst-change-snafu-shows-stone-age-routines

[5] /2008/01/23/modified-version-of-twitterlex/

[6] http://erik.range-it.de/wordpress/plugins/syntaxhighlighter/

[7] syntaxphp.zip

[8] syntax.diff

[9] http://www.gnu.org/philosophy/free-sw.html

[10] /2007/06/19/richard-stallman/

Democracy

Adrian Kosmaczewski

2008-01-30

What amazes me most about democracy is that:

- Most people associate it with just “voting”. We can vote, hence we live in a free country.

- There is a clear association in our minds between power and corruption. Democracy is not the exception here.

Why do we vote, then? What for? Has “modern democracy” really changed anything in this world in the last 200 years? We still have slavery, genocides and hunger (and even more now than before). Whether you see them or not depends on the sources of information you read.

Don’t get me wrong. I’m not asking for a dictatorship; I think that democracy is still the best deal you can get from all other systems. At least you can have the **hope** of getting rid of your current dictator after the term is over. But don’t fool yourself, you don’t get much more than hope. The history is full of “democracies” with the same president being voted again and again through the decades.

I’m asking for a collective scream.

I think it is possible to behave like adults and care about each other, at last, without the “powers that be” watching over your shoulder and eating 70% of what you create, whether you like or not.

I care. Do you?

6 blogs you should read... absolutely

Adrian Kosmaczewski

2008-01-31

Maybe you don't have the time or will to read those damn 6 books every year[1]. And maybe that's fine (for you). I will give you a list of 6 blogs that you can add to your RSS reader. If you care a little about software engineering, new technology trends, and enjoy reading well-written, usually lengthy articles (as I do), IMHO you should be checking these.

No, mine isn't in the list :) You can proceed without fear now.

1. Paul Graham[2]: Paul founded Viaweb in the mid 90s, and later sold it to become Yahoo! Stores[3]. He then created Y Combinator[4], a successful venture capital firm. His writings are, to say the least, controversial (you usually agree or not with him, but he never leaves you indifferent). I cannot help thinking that his points are excellent. I also enjoyed his Hackers and Painters book[5]; he is an excellent writer and loves to share his knowledge, both about programming and startups.
2. Steve Yegge[6]: he works at Google, and lately has got some attention thanks to his Rhino on Rails[7] project. He has been interviewed by Dion Almaer[8] (of Ajaxian[9] fame) and the video is on YouTube[10]. His blog is really interesting. His are not just "rants", as the title says; it's first hand experiences, funny and oh so true about our daily world of software engineering.
3. Reg Braithwaite[11]: I discovered Reg's blog while writing my now legendary Erlang post[12] last year. His articles have a distinct human touch, and I enjoy every word in them. I cannot but recommend his blog.

[1] /blog/best-books-of-2007/
[2] http://www.paulgraham.com/index.html
[3] http://smallbusiness.yahoo.com/ecommerce/
[4] http://ycombinator.com/
[5] http://www.paulgraham.com/hp.html
[6] http://steve-yegge.blogspot.com/
[7] http://steve-yegge.blogspot.com/2007/06/rhino-on-rails.html
[8] http://almaer.com/blog/
[9] http://ajaxian.com/
[10] http://www.youtube.com/watch?v=1QD9XQm_Jd4
[11] http://weblog.raganwald.com/
[12] /blog/erlang/

4. Steve McConnell[13]: he is the author of “Code Complete”, a major landmark book in the history of software engineering, and his “10x Software Development” blog is a natural continuation to that book. This blog is full of advice, not only for pure coding stuff, but also about project management and estimation tasks.
5. Martin Fowler[14]: his “bliki” (a cross between a blog and a wiki) is, well, an absolute reference. I do not have to present him. You should have read everything he’s written so far by now. You must have, actually.
6. Scott Berkun[15]: not strictly about software engineering, but rather about project management. Scott knows his stuff, and he knows how to explain it. If you care about your projects, you should hear what he says.

Finally, Joel Spolsky[16] also deserves to be listed, but in my opinion, his best articles are those from the 2001-2003 era. He is more concentrated in his company[17] now, and so he still writes new, interesting stuff, but not at the same rate. You might want to check it out by yourself! The archives are plenty of treasures worth reading.

A couple more? OK! Here you go: Coding Horror[18], Yariv Sadan[19] (about Erlang), Scott Stevenson[20] (about Cocoa), Presentation Zen[21] (about… well, presentations), and finally the O’Reilly Beautiful Code[22] blog.

Finally there were much more than 6, but hey, happy reading anyway! Feel free to leave me your preferred ones in the comments below.

[13] http://blogs.construx.com/blogs/stevemcc/default.aspx
[14] http://martinfowler.com/bliki/
[15] http://www.scottberkun.com/
[16] http://www.joelonsoftware.com/
[17] http://fogcreek.com/
[18] http://www.codinghorror.com/blog/
[19] http://yarivsblog.com/
[20] http://theocacao.com/
[21] http://www.presentationzen.com/
[22] http://beautifulcode.oreillynet.com/

Sistema Propano

Adrian Kosmaczewski

2008-02-17

Back in 2004 I drafted a content management system based on PHP and AJAX, featuring ideas borrowed from everywhere and that, all in all, formed a rather coherent set. I wrote the system in a couple of days, in my daily commuting between Lausanne and Geneva, and it was called **Propano**.

The trigger was an article about the XMLHttpRequest object[1] that appeared in the Apple Developer Connection site. During my work at SoftPlumbers in 2002 I had used it in our flagship application (which could only work on Explorer 5.5 and above, using this component), and I remember being so impressed about having a web page updating itself without a whole reloading, that when I saw that you could do the same on Safari and Firefox I thought, this is really big. And it was. A couple of months later this way of doing things got the name AJAX[2] and the rest is history.

I should have published this code before, but heck, better late than never :)

I wanted to write an enhanced version of my original "tribute to NeXT" page, that I had as a homepage from 2000 to 2004; as far as I can remember, I had the only "DHTML" windowing application that you could use in Netscape 4 (!); it used the dreadful <LAYER> tag, and it used the famous <IFRAME> trick to get information from the server without reloading the page.Ahhh, that was hacking, really. Now all of this is oh so much[3] standardized[4], it makes me see those days with nostalgy. Well, almost :)

With Propano you can't do much (it's just a draft, remember) but you can go to the menu "Archivo / Abrir" ("File / Open") and you'll make an AJAX call to the server. Big deal, huh? You can select "Ventanas / Crear Muchas" ("Window / Create Many") and open lots of windows. You can resize windows using the status bar. You can close, minimize and maximize windows (you'll need to double-click when clicking icons!) . Finally, you can drag the menu around, open items, etc. However,

[1] http://developer.apple.com/internet/webcontent/xmlhttpreq.html
[2] http://www.adaptivepath.com/ideas/essays/archives/000385.php
[3] http://jquery.com/
[4] http://prototypejs.org/

I’ve never used the NeXT system, and this is the closer I could get by reading the design documents that I found here and there.

The web was created on a NeXT workstation.[5] This is a humble way to close the circle, if you want. It works on Safari, Firefox and it should work fine but it does not work with Opera 9; actually it should also work on Internet Explorer 6, but I haven’t tested it in that browser for years… The NeXTClock Applet comes from here[6] and I was using it in my previous <LAYER> + <IFRAME> attempt.

Anyway, for those interested, here’s the code[7]. Feel free to play with it!

[5] http://www.w3.org/People/Berners-Lee/WorldWideWeb.html
[6] http://the-labs.com/NeXTClock/
[7] propano.zip

How to Count Words in Latex Files?

Adrian Kosmaczewski

2008-03-01

I am a big LaTeX[1] fan, mostly thanks to my friend Cedric[2] who introduced me to it ;) And I don't regret it at all; there is simply no better way to create long, beautiful PDF documents, particularly during these times of dissertation writing! I'm in my last step towards the Master's degree I've been working on for the last two years, and creating documents is an important part of that.

LaTeX works for me, because:

- It's cross-platform (and I need that for my project!);
- It's text based (I can edit the files with any decent editor; personally I use and TexShop[3] and sometimes TextMate[4]);
- I can generate PDF, plain text, RTF, and much more from the same source;
- I can split my documents in several others and work separately in each;
- I can generate meaningful diffs using Subversion (to see what I've changed in every revision);
- I can manage the bibliography for my papers easily (using the awesome BibDesk[5] tool);
- I don't have to cope with a buggy text editor that crashes every so often!
- I can generate gorgeous, absolutely beautiful documents. Easily.

For my last document, the dissertation, I have a numeric limit in the number of words (~ 10K to 15K words) and I need to count the number of words in the documents I generate. Since I'm not using Word, nor KOffice nor OpenOffice, this simple requirement becomes more complex to fulfill. But working in a Unix environment has its benefits; first I found this solution[6]:

```
$ detex file.tex | wc -w
```

This command provides a first approach to the problem; however, it

[1] http://www.latex-project.org/
[2] http://www.linkedin.com/in/cducommun
[3] http://www.uoregon.edu/~koch/texshop/
[4] http://macromates.com/
[5] http://bibdesk.sourceforge.net/
[6] http://www.tex.ac.uk/cgi-bin/texfaq2html?label=wordcount

just strips off the LaTeX commands, even those that generate content in the final document. For example, if you have a macro that puts in bold the name of your project, those words will not appear in the final calculation even if they do appear in the final document. Clearly not acceptable. Googling a bit more, I found what I was looking for[7]:

```
$ ps2ascii file.pdf | wc -w
```

In this case we're working on the final PDF document, and of course the final result is much, much more interesting.

Happy typesetting!

[7] http://markelikalderon.com/blog/2006/11/04/latex-word-count-and-textmate/

Barcamp Lausanne 2

Adrian Kosmaczewski

2008-03-04

Just a quick post to announce that I will be speaking next Saturday at the second Barcamp Lausanne[1]. My topic?

> What happened to those nice words called “Software Quality”? A grumpy developer’s perspective.

Feel free to come! I’d love to meet you there.

[1] http://barcamp.ch/BarCampLausanne2

iPhone SDK: Une Nouvelle Ere Démarre

Adrian Kosmaczewski

2008-03-07

Il y a de moments clés dans l'histoire de la technologie. Hier soir, vers 18h (heure suisse), il s'est produit l'un de ces moments. Apple a dévoilé un SDK (Software Development Kit) pour l'iPhone, et le monde du développement logiciel mobile ne sera plus jamais le même. Voici pourquoi.

Après de multiples rumeurs, Apple a finalement devoilé hier soir un SDK (Software Development Kit) pour son téléphone fétiche, l'iPhone. Cette nouvelle, bien que apparemment sans intérêt pour l'utilisateur moyen, aura des effets sans précédents, autant pour chacun de nous, communs mortels utilisateurs de téléphonie mobile, comme pour l'industrie toute entière.

En effet, bien que ce n'est pas la première fois qu'un fabriquant de téléphones mobiles offre un tel produit, l'engouement autour de l'iPhone tout comme ses charactéristiques techniques font que la nouvelle prenne une toute autre ampleur.

Pour ceux qui ne le sauraient pas, un SDK est un ensemble d'outils, qui permet aux développeurs de logiciels de pouvoir créer des applications autour d'une plate-forme. Par exemple, J2EE ou Ruby on Rails peuvent être considérés comme des SDKs, spécifiquement conçus pour créer des sites web, qui stockent leurs informations dans un système de base de données structuré. De la même façon, chaque système d'exploitation comme Windows, Mac OS X, Linux, mais aussi Solaris, QNX, BSD et n'importe quel autre, est généralement fourni avec un SDK (usuellement gratuit) pour que les développeurs puissent augmenter les capacités de la plate-forme, en l'étendant dans des façons nouvelles et inconnues par son fabriquant.

Dans les cas des téléphones portables, aucun des SDK existants (la plupart basés sur le langage Java) n'ont eu un succès fulgurant, bien que le hardware utilisé pour le monde mobile devient chaque jour plus puissant, et bien que l'expérience utilisateur de la plupart de téléphones laisse vraiment à désirer. Quiconque aura attendu 2 minutes pour que son jeu puisse être utilisé sur son portable (le temps pour que le logo "Java" disparaisse), ou quiconque aura vu son SMS disparaître lorsque le téléphone redémarre inopinément, saura de quoi je parle.

Voici donc Apple; une société dont l'iPhone est la deuxième intervention sérieuse dans le monde de l'informatique mobile, le premier essai, le Newton, s'étant soldé sur un échec commercial (mais un succès conceptuel, comme Palm l'a prouvé quelques années plus tard). Cette fois-ci, la société fondée par Steve Jobs en 1976 compte bien changer les règles du jeu, et le SDK annoncé hier soir est une partie fondamentale de cette stratégie.

D'un point de vue technique, on peut dire sans se tromper que l'iPhone et un Mac de poche. Le système d'exploitation de l'iPhone (ou "iPhone OS") est une version miniature du Mac OS X, le software qui gère le fonctionnement de n'importe quel Mac sur le marché. Mac OS X compte une panoplie complète de librairies et de fonctionnalités prêtes à l'emploi, la plupart d'entre elles développées et testées continuellement depuis la fin des années 80 (à l'époque de l'ordinateur NeXT). Tout cela est maintenant à disposition des développeurs dans l'iPhone OS.

Mais ce n'est pas seulement un compilateur qu'Apple fournit avec son SDK; c'est aussi une suite d'utilitaires qui permet de créer le code et de le corriger (Xcode), de créer graphiquement des interfaces utilisateurs avec le moindre effort (Interface Builder), de voir son exécution et de paramétriser ses performances (Instruments) et de livrer les applications aux utilisateurs (App Store).

Finalement, Apple s'est inspiré de Cocoa, la librairie et runtime graphique utilisé dans le Mac, en ajoutant les contrôles nécessaires pour créer des applications iPhone, qui gèrent correctement les actions de l'utilisateur, lorsqu'il promène ses doigts sur le "touch screen" de l'appareil. Cette nouvelle version de Cocoa, "Cocoa Touch" utilise tout le pouvoir d'Objective-C, le langage de programmation orienté objet, vraie "lingua franca" du développement sur Mac.

Objective-C est un langage unique en son genre: c'est probablement le seul langage de programmation dynamique et compilé à la fois; il offre toute la puissance et vitesse du langage C, avec la beauté et la grâce de la programmation objet, tout en fournissant un environnement qui se prête au "développement rapide" d'applications comme aux plus hautes performances.

Bref, l'iPhone OS ouvre la porte à une nouvelle génération d'applications mobiles: vitesse native, support pour "multithreading", rapidité de création, facilité de maintenance et de déploiement,

et accès natif aux multiples capacités de l'iPhone (caméra, carnet d'adresses, navigateur web intégré, système de géolocalisation, et j'en passe). Je vous invite à voir la vidéo de la présentation d'hier pour voir les capacités de l'outil; avancez jusqu'a la minute 40, et regardez ce qu'on peut faire avec.

L'iPhone SDK est disponible gratuitement (il fait 2 GB!) chez http://developer.apple.com (il est juste nécéssaire de créer un compte ADC - Apple Developer Connection -, ce qui est gratuit et ne prend que quelques secondes). Dans la version disponible actuellement, seul l'Interface Builder fait défaut, mais les autres outils sont présents et prêts à l'emploi. La version définitive sera offerte dès le mois de juin prochain.

Entre temps, pour les développeurs qui voudraient apprendre Cocoa, je vous recommande trois livres:

- "Programming in Objective-C" par Stephen Kochan (ISBN 978-0672325861)
- Learning Cocoa with Objective-C, Second Edition, par James Duncan Davidson (ISBN 978-0596003012)
- Cocoa Programming for Mac OS X, par Aaron Hillegass (ISBN 978-0321213143)

Et quatre websites:

- iPhone Dev Center
- Cocoa Dev Central- CocoaDev
- Theocacao, by Scott Stevenson

Je reste aussi à votre disposition pour toute question à propos de Cocoa, Objective-C et des technologies Mac, ayant utilisé Cocoa (avec un énorme plaisir!) depuis 2002.

Happy coding!

Null References

Adrian Kosmaczewski

2008-03-07

There's an interesting[1] discussion[2] going on[3] these days on Ruby blogs about, basically, how to avoid one of the most common, annoying, easy-to-create bugs in any programming language: **calling a method on a null reference (or pointer, depending on your language).**

[1] http://ruby.tie-rack.org/53/a-better-try-chaining-methods-and-nil/

[2] http://chalain.livejournal.com/69460.html

[3] http://blog.rubyenrails.nl/articles/2008/02/29/our-daily-method-18-nilclass-method_missing

[4]

This single issue happens all the time, in garbage-collected and non-managed languages, static and dynamic, weakly and strongly typed; you have a handler variable “pointing” to an object, and before calling any methods on it, you’d better be sure that the object is there; you end up using assertions, “if” statements (and all of its variants), boilerplate code all over the place, when everything you want to do is to call that damn method. It’s frustrating, time-consuming and oh so common that we just try to not to think about it anymore.

In Objective-C there is an easy solution to this problem: you can safely send messages to (which is roughly equivalent to “call methods on”) nil[5], but of course, not everyone likes that[6]. I think that this single feature is responsible for a big deal of “user perceived stability” in the whole Cocoa runtime; it exchanges a what could be a potentially fatal, low-level and unrecoverable error (leading to a complete application crash) into a purely functional one; “look, I’ve clicked here and nothing happens!”. The application does not crash anymore, it just does not do what it should, because the object that should have received the

[4] https://www.flickr.com/photos/oddwick/160592075/

[5] http://www.osnews.com/permalink?f94413

[6] http://www.koziarski.net/archives/2007/1/22/cocoa-and-objective-c

message is not there. The user has a smoother experience, and this means a lot in the long term.

The beauty here, shared by Objective-C and Ruby (and as far as I understand, Smalltalk and other languages), is that messages and method implementations are decoupled; you can forward messages from one object to another, creating chains of responsibility[7]; you can log messages before you execute them (doing some aspect-oriented stuff without all the marketing fuss); you can change the implementation (or even remove it and place it somewhere else altogether) without breaking your clients. It brings a whole lot of power, with the small overhead of having a runtime process dispatching methods, which is, yes, takes slightly longer than a virtual method call, and (of course) even longer than compile-time bound method call.

I think that dynamic languages have a definitive advantage in this field, and this is why I prefer them in environments with requirements evolving constantly, where clients more often than not request new features and where you must reduce maintenance costs; not having your app crash in your face is a good sign of software, and languages that allow you to deliver them are fundamental.

[7] http://developer.apple.com/documentation/Cocoa/Conceptual/CocoaFundamentals/CocoaDesignPatterns/chapter_5_section_3.html#//apple_ref/doc/uid/TP40002974-CH6-SW25

Blow Your Mind

Adrian Kosmaczewski

2008-03-11

Take a careful look at this:

```
#include <iostream>

class Gadget
{
public:
    void sayHello() const
    {
        std::cout << "Gadget!" << std::endl;
    }
};

class Widget
{
public:
    void sayHello() const
    {
        std::cout << "Widget!" << std::endl;
    }
};

template <class T>
class OpNewCreator
{
public:
    T* create()
    {
        std::cout << "Using 'new': ";
        return new T;
    }
};

template <class T>
class MallocCreator
{
public:
    T* create()
```

```
    {
        std::cout << "Using 'malloc': ";
        void* buf = std::malloc(sizeof(T));
        if (!buf) return 0;
        return new(buf) T;
    }
};

template <class T, class B>
class Creator : public T<B>
{
public:
    void exec()
    {
        B* obj = this->create();
        obj->sayHello();
        delete obj;
    }
};

typedef Creator<MallocCreator, Widget> Manager;

int main (int argc, char * const argv[])
{
    Manager obj;
    obj.exec();
    return 0;
}
```

Try changing `MallocCreator` by `OpNewCreator` and `Widget` by `Gadget` in the typedef of line 57, recompile and run; of course you can provide default values, so that

```
template <class T = MallocCreator, class B = Widget>
class Creator : public T<B>
```

so that you just do

```
typedef Creator<>; Manager;
```

I've just started reading "Modern C++ Design" by Andrei Alexandrescu and I've already my head spinning out of orbit. This is amazing (and by giving a quick look at the rest of the book, there's even more incredible stuff there)!

Templates

Adrian Kosmaczewski

2008-03-13

Did you knew this is possible in C++? I didn't.

```
void Fun()
{
    class Local
    {
        //... member variables ...
        //... member functions ...
    };

    // ... code using Local ...
}
```

This feature is called "local classes" and is part of the standard; the limitations are that these local classes cannot have static member variables and cannot access nonstatic local variables. But, as Alexandrescu points out (page 28), you can use them in template functions:

```
class Interface
{
public:
    virtual void Fun() = 0;
    // ...
};

template <class T, class P>
Interface* makeAdapter(const T& obj, const P& arg)
{
    class Local : public Interface
    {
    public:
        Local(const T& obj, const P& arg)
            : obj_(obj), arg_(arg) {}

        virtual void Fun()
        {
            obj_.Call(arg_);
        }
```

```
    private:
        T obj_;
        P arg_;
    };

    return new Local(obj, arg);
}
```

Not only that, but partial template specialization and template-based compile-time verifications just blew me away. The second chapter of the book was realizing I just haven't had the slightest clue about all the cool things you could do with C++!

Screen Savers for the Mac Using Flash

Adrian Kosmaczewski

2008-03-18

This is an evening project that turned out to be really cool. Let's say that you're a Macromedia Flash designer, and your clients ask you to bundle your nice Flash movie as a screen saver. What to do? For Windows there are free utilities to convert a SWF into a screen saver, but not for the Mac - and the first commercial one costs around USD 200!

I propose here a simple solution for this problem:

1. Using Xcode, you can create screen savers (basically, Cocoa apps).
2. Cocoa applications can host a WebKit component (basically, Safari).
3. Safari can show local HTML pages (basically, `file:///` stuff).
4. And HTML pages can show Flash movies (basically, `<OBJECT>` and `<EMBED>`)

The idea, then, is to bundle your own page, with your own movie, inside the screen saver bundle, and show the Flash movie this way. Easy said, easy done.

It all goes nice until... you try this solution :) The problem is, Flash movies loaded from `file:///` URLs are blocked by the built-in security mechanisms of Adobe... and you have to find a workaround for that. Mine was to follow the instructions[1] on how to bypass the Flash security mechanism, and then create an installer package that will do what's needed for you to enjoy the screen saver.

You can get both the project and the installer package in my projects section[2] (source[3]). I've tested the installer in three different machines, and it worked, so I hope it'll work for you too :) Enjoy! As always, try this at your own risk. Murphy says that things can go wrong, so watch out.

[1] http://kb.adobe.com/selfservice/viewContent.do?externalId=1165eb90
[2] installer.zip
[3] project.zip

Playing With HTTP Libraries

Adrian Kosmaczewski

2008-03-26

It's fun to find out how to tackle the same task in different programming languages; in this case, it's all about doing HTTP requests over a network: fortunately, there are networking libraries in virtually all major programming languages. In my current project, I'm generating wrappers easing the access to the core of the project itself, a RESTful API. This way, developers interested in using the API can just take a wrapper, include it in their projects, and start coding right away. No need to know this (relatively low-level) stuff; just use the API. The wrappers themselves are auto-generated from the API definition itself, but that's another story ;)

Below there is a sample of the different ways I've found to do a network access to a remote server, using HTTP Basic Authentication and a couple of headers, in PHP, Ruby, Python, JavaScript, and even Objective-C! I'm even generating ActionScript 3.0 code, but I'm not a Flash coder :) So I'll post the wrappers that work best at the moment, and in the future I'll include other examples, particularly for .NET, C++ and Java.

In all the cases below, there is a "request" function or method that takes an HTTP verb (GET, POST, PUT, DELETE, etc), a URL (without the slash "/" at the beginning) and some parameter data, in the form of a dictionary. The function wraps the underlying libraries of each programming language, offering a simpler interface, and allowing for HTTP Basic Authentication (for HTTP Digest Authentication it would be much, much more complex!). There are synchronous (useful for server or command-line applications) and asynchronous versions (for GUI systems). Off to the code!

In PHP (synchronous):

```
$conf = array(
    "server" => "localhost",
    "username" => "",
    "password" => ""
);

function request($verb, $url, $parameters = array()) {
    global $conf;
    $headers = array(
```

```
    "Content-Type: text/html; charset=utf-8",
    "Accept: application/javascript",
    "Cache-Control: no-cache",
    "Pragma: no-cache",
    "Content-length: " . strlen($xml_data),
    "Authorization: Basic " . base64_encode($conf["username"] . ":" . $conf["pa
    );

    $path = $conf["server"] . "/" . $url;
    $conn = curl_init();
    curl_setopt($conn, CURLOPT_URL, $path);
    curl_setopt($conn, CURLOPT_HTTPHEADER, $headers);
    curl_setopt($conn, CURLOPT_USERAGENT, "Identify yourself!");
    curl_setopt($conn, CURLOPT_CUSTOMREQUEST, strtoupper($verb));
    curl_setopt($conn, CURLOPT_POSTFIELDS, $xml_data);
    curl_setopt($conn, CURLOPT_RETURNTRANSFER, 1);
    $data = curl_exec($conn);
    $code = curl_getinfo($conn, CURLINFO_HTTP_CODE);

    if($code == 200 && strtoupper($verb) == "GET") {
    return json_decode($data);
    }
    else {
    return $data;
    }

    curl_close($conn);

    return $data;
}
```

The curl library has a distinctive C smell :) The good thing is that after doing this wrapper, doing the C++ one will be fairly straightforward!

In Ruby (synchronous):

```
require 'net/http'
require 'uri'
require 'json'

$conf = {
    "server" => "localhost",
    "username" => "",
    "password" => ""
}

def request(verb, url, parameters = nil)
    Net::HTTP.start($conf["server"]) do |http|
    headers = {
        "Accept" => "application/javascript",
        "User-Agent" => "Identify yourself!"
```

```
    }
    path = "/" + url
    req = nil
    case verb.upcase
        when "GET":
        req = Net::HTTP::Get.new(path, headers)
        when "POST":
        req = Net::HTTP::Post.new(path, headers)
        when "PUT":
        req = Net::HTTP::Put.new(path, headers)
        when "DELETE":
        req = Net::HTTP::Delete.new(path, headers)
        else
        raise Exception.new("Invalid HTTP verb")
    end
    req.basic_auth $conf["username"], $conf["password"]
    req.set_form_data(parameters) if not parameters == nil
    response = http.request(req)
    if response.code == "200" && verb.upcase == "GET"
        JSON.parse(response.body)
    else
        print response.code + " " + response.message + " " + response.body
    end
    end
end
```

The Ruby library is the only one I found so far that features the four HTTP verbs in the interface!

In Python (synchronous):

```
import simplejson
import httplib, urllib
import base64

conf = {
    "server": "localhost",
    "username": "",
    "password": "",
}

def request(verb, url, parameters = {}):
    params = urllib.urlencode(parameters)
    base64string = base64.encodestring('%s:%s' % (conf["username"], conf["passw
    headers = {
        "Accept": "application/javascript",
        "Authorization": "Basic %s" % base64string,
        "User-Agent": "Identify yourself!",
    }
    conn = httplib.HTTPConnection(conf["server"])
    conn.request(verb.upper(), "/" + url, params, headers)
```

```
    response = conn.getresponse()
    if response.status == 200 and verb.upper() == "GET":
        obj = simplejson.loads(response.read())
        return obj
    else:
        print response.status, response.reason, response.read()
    conn.close()
```

I much prefer the Ruby way, if you ask me. I don't know, it's more elegant. I can't get used to the indenting!

In JavaScript, using Prototype (asynchronous):

```
var conf = {
    server: "localhost",
    username: "",
    password: ""
};
var request = function(verb, url, parameters, successHandler, errorHandler) {
    // A really ugly way to do HTTP Basic Auth, I know :)
    var api_url = ["http://", conf.username.replace(/@/, "%40"), ":", conf.pass

    new Ajax.Request(api_url, {
        method: verb.toLowerCase(),
        parameters: parameters,
        requestHeaders: {
            "Accept": "application/javascript",
            "Cache-Control": "no-cache",
            "Pragma": "no-cache",
            "Content-Type": "text/html; charset=utf-8"
        },
        onSuccess: function(transport) {
            if(successHandler) successHandler(transport);
        },
        onFailure: function(transport) {
            if(errorHandler) errorHandler(transport);
        }
    });

};
```

Again in JavaScript, but this time using jQuery (asynchronous):

```
var conf = {
    server: "localhost",
    username: "",
    password: ""
};
var request = function(verb, url, parameters, successHandler, errorHandler) {
    var api_url = ["http://", conf.username.replace(/@/, "%40"), ":", conf.pass

    $.ajax({
```

```
        type: verb.toUpperCase(),
        url: api_url,
        async: true,
        data: parameters,
        cache: false,
        // jQuery does not allow to change headers otherwise (?) than using the
        beforeSend: function(xhr) {
            xhr.setRequestHeader("Content-Type", "text/html; charset=utf-8");
            xhr.setRequestHeader('Accept', 'application/javascript');
            xhr.setRequestHeader('Cache-Control', 'no-cache');
            xhr.setRequestHeader('Pragma', 'no-cache');
        },
        complete: function(engine, textStatus) {
            if (engine.readyState == 4) {
                if (engine.status == 200 || engine.status == 201) { if(successH
                else { if(errorHandler) errorHandler(engine); }
            }
        }
    });

};
```

The last JavaScript one, but this time using bare bones XMLHttpRequest (asynchronous):

```
var conf = {
    server: "localhost",
    username: "",
    password: ""
};
var request = function(verb, url, parameters, successHandler, errorHandler) {
    var api_url = ["http://", conf.username.replace(/@/, "%40"), ":", conf.pass

    var params = [];
    for(var item in parameters) {
        params.push(escape(item) + "=" + escape(parameters[item]));
    }
    var engine = null;
    if (window.XMLHttpRequest) {
        engine = new XMLHttpRequest();
    }
    if (window.ActiveXObject) {
        engine = new ActiveXObject("Microsoft.XMLHTTP");
    }
    engine.onreadystatechange = function stateChange() {
        if (engine.readyState == 4) {
            if (engine.status == 200 || engine.status == 201) { if(successHandl
            else { if(errorHandler) errorHandler(engine); }
        }
    };
    engine.open(verb.toUpperCase(), api_url, true);
```

```
    // setRequestHeader() must be called AFTER open() !!
    engine.setRequestHeader("Content-Type", "text/html; charset=utf-8");
    engine.setRequestHeader('Accept', 'application/javascript');
    engine.setRequestHeader('Cache-Control', 'no-cache');
    engine.setRequestHeader('Pragma', 'no-cache');
    engine.send(params.join("&"));
};
```

To use these JavaScript versions, just pass a couple of functions as parameters, and you're done; they will be called in case of success or error, asynchronously.

Finally, the Cocoa / Objective-C wrapper (both synchronous and asynchronous, header and implementation files):

```
/*
Command-line example that uses this library:
#import <Foundation/Foundation.h>

int main (int argc, const char * argv[]) {
    NSAutoreleasePool * pool;
    pool = [[NSAutoreleasePool alloc] init];

    AKResourceSubClass* resource;
    resource = [[AKResourceSubClass alloc] init];
    [resource get];
    NSDictionary* plist = [resource getPropertyList];
    // Do something with the results...

    [pool drain];
    return 0;
}
*/

#import <Foundation/Foundation.h>

@interface AKResource : NSObject {
    NSMutableData* receivedData;

    NSString* mimeType;
    NSString* server;
    NSString* username;
    NSString* password;

    BOOL async;
}

-(id)init;
-(void)setServer:(NSString*)serverName;
-(void)setUsername:(NSString*)user
    andPassword:(NSString*)pwd;
```

```
-(void)connection:(NSURLConnection *)connection
    didReceiveResponse:(NSURLResponse *)response;
-(void)connection:(NSURLConnection *)connection
    didReceiveData:(NSData *)data;
-(void)connection:(NSURLConnection *)connection
    didFailWithError:(NSError *)error;
-(void)connectionDidFinishLoading:(NSURLConnection *)connection;
-(void)sendRequestTo:(NSString*)urlString
    usingVerb:(NSString*)verb
        withParameters:(NSDictionary*)parameters;
-(void)setAsynchronous:(BOOL)asynchronously;
-(NSDictionary*)getPropertyList;
-(NSString*)getResponseText;

@end
```

Now the implementation file:

```
#import "AKResource.h"

@implementation AKResource

-(id)init
{
    self = [super init];

    if(self)
    {
        receivedData = [[NSMutableData alloc] init];

        mimeType = @"application/plist";
        server = @"localhost";
        username = @"";
        password = @"";

        async = NO;
    }

    return self;
}

-(void)setServer:(NSString*)serverName
{
    server = serverName;
}

-(void)setUsername:(NSString*)user andPassword:(NSString*)pwd
{
    username = user;
    password = pwd;
}
```

```
-(void)sendRequestTo:(NSString*)resource
    usingVerb:(NSString*)verb
    withParameters:(NSDictionary*)parameters
{
    NSURL* url = [NSURL URLWithString:[NSString
        stringWithFormat:@"http://%@/%@", server, resource]];

    NSMutableDictionary* headers = [[NSMutableDictionary alloc] init];
    [headers setValue:@"text/html; charset=utf-8"
        forKey:@"Content-Type"];
    [headers setValue:mimeType forKey:@"Accept"];
    [headers setValue:@"no-cache" forKey:@"Cache-Control"];
    [headers setValue:@"no-cache" forKey:@"Pragma"];

    NSMutableURLRequest* request;
    request = [NSMutableURLRequest requestWithURL:url
                cachePolicy:NSURLRequestUseProtocolCachePolicy
                timeoutInterval:60.0];
    [request setHTTPMethod:verb];
    [request setAllHTTPHeaderFields:headers];

    if (parameters)
    {
        NSMutableString* params = [[NSMutableString alloc] init];
        for (id key in parameters)
        {
            [params appendFormat:@"%@=%@&",
                [key stringByAddingPercentEscapesUsingEncoding:NSUTF8StringEnco
                [[parameters objectForKey:key]
                    stringByAddingPercentEscapesUsingEncoding:NSUTF8StringEncod
        }
        [params deleteCharactersInRange:NSMakeRange([params length] - 1, 1)];

        NSData* body = [params dataUsingEncoding:NSUTF8StringEncoding];
        [request setHTTPBody:body];
    }
    if (async)
    {
        NSURLConnection* connection;
        connection = [[NSURLConnection alloc]  initWithRequest:request
                                                delegate:self
                                                startImmediately:YES];

        if (!connection)
        {
            NSLog(@"Could not open connection to resource");
        }
    }
    else
```

```
    {
        NSURLResponse* response = [[NSURLResponse alloc] init];
        NSError* error = [[NSError alloc] init];
        NSData* data = [NSURLConnection
            sendSynchronousRequest:request
            returningResponse:&response
            error:&error];
        [receivedData setData:data];
    }
}

-(void)connection:(NSURLConnection *)connection
        didReceiveAuthenticationChallenge:(NSURLAuthenticationChallenge *)chall
{
    NSURLCredential *newCredential;
    newCredential = [NSURLCredential credentialWithUser:username
                        password:password
                        persistence:NSURLCredentialPersistenceNone];
    [[challenge sender] useCredential:newCredential
        forAuthenticationChallenge:challenge];
}

-(void)connection:(NSURLConnection *)connection
    didReceiveResponse:(NSURLResponse *)response
{
    NSHTTPURLResponse* httpResponse;
    httpResponse = (NSHTTPURLResponse*)response;
    int statusCode = [httpResponse statusCode];
    if (statusCode != 200)
    {
        NSMutableDictionary* info;
        info = [NSMutableDictionary dictionaryWithObject:[response URL]
            forKey:NSErrorFailingURLStringKey];
        [info setObject:@"An error code different than 200 was received!"
            forKey:NSLocalizedDescriptionKey];
        NSError* error = [NSError errorWithDomain:@"API Wrapper"
            code:statusCode
            userInfo:info];
        NSDictionary* errorData = [NSDictionary
            dictionaryWithObject:error
            forKey:@"error"];

        [[NSNotificationCenter defaultCenter]
            postNotificationName:@"ConnectionDidFailWithStatusCodeNotOK"
            object:self
            userInfo:errorData];
    }
    [receivedData setLength:0];
}
```

```
-(void)connection:(NSURLConnection *)connection
    didReceiveData:(NSData *)data
{
    [receivedData appendData:data];
}

-(void)connection:(NSURLConnection *)connection
    didFailWithError:(NSError *)error
{
    [connection release];

    NSDictionary* errorData = [NSDictionary
                                dictionaryWithObject:error
                                forKey:@"error"];

    [[NSNotificationCenter defaultCenter]
        postNotificationName:@"ConnectionDidFailWithError"
        object:self
        userInfo:errorData];

    NSLog(@"Connection failed! Error - %@ %@",
            [error localizedDescription],
            [[error userInfo] objectForKey:NSErrorFailingURLStringKey]);
}

-(void)connectionDidFinishLoading:(NSURLConnection *)connection
{
    [[NSNotificationCenter defaultCenter]
        postNotificationName:@"ConnectionDidFinishLoading"
        object:self];
    [connection release];
}

-(void)setAsynchronous:(BOOL)asynchronously
{
    async = asynchronously;
}

-(NSDictionary*)getPropertyList
{
    NSString* errorStr = nil;
    NSDictionary* propertyList;
    NSPropertyListFormat format;

    propertyList = [NSPropertyListSerialization
                    propertyListFromData:receivedData
                    mutabilityOption: NSPropertyListImmutable
                    format: &format
                    errorDescription: &errorStr];
    return propertyList;
```

```
}

-(NSString*)getResponseText
{
    return [[NSString alloc]
            initWithData:receivedData
            encoding:NSUTF8StringEncoding];
}

@end
```

Rather verbose this last one huh? Objective-C has a distinctive characteristic, inherited from Smalltalk (I suppose): methods with named parameters, like `stringByAddingPercentEscapesUsingEncoding:NSUTF8StringEncoding` (decidedly, one of the longest method calls ever :) The good thing is that the code reads like a book. This is why I like this language, as well as Ruby: you can read code in these languages very easily.

Another nice aspect of this Objective-C wrapper is that if you have an API that can return "Property Lists" (like the one I've been working on lately), you can send object graphs serialized in XML that are completely Cocoa-compatible. That's what the getPropertyList method does: client code can deal with standard NSDictionary instances, no need to do anything else!

Of course these are not the only ways to do this, but so far these wrappers have helped me a lot. Don't hesitate to leave your comments below! And most important, have fun! As always, use this code at your own risk.

WWDC 2008: I'll Be There!

Adrian Kosmaczewski

2008-03-29

I've just bought my e-ticket for Apple's Worldwide Developer Conference 2008! This one will be the first one featuring iPhone developer tracks, and that's one of the main reasons for me to go. Another good one is that neither Claudia nor me know San Francisco, so it'll be a great time to hang out and visit the city.

Will you be there? Feel free to leave me a comment and we'll meet at San Francisco, from June 9th to 13th!

Update, 2008-06-09: promise kept[1].

[1] /blog/i-was-there/

Django Architecture Approaches

Adrian Kosmaczewski

2008-04-04

I've just had a very interesting conversation with my colleague Marco[1] about different approaches to the organization of code inside a Django application.

As you might know (and if you don't I'll tell you anyway), Django's views (somehow occupying the "Controller" level in an MVC architecture) must take (at least) an HttpRequest[2] instance as a parameter and must return an HttpResponse[3] instance. **That's how it goes in Django, this is the law[4].** This means that you must be sure that the last instruction in your request processing code (in whichever way you've organized it) must return an HttpResponse instance, usually calling the HttpResponse() constructor (or of any of its useful subclasses), or by calling the `django.shortcuts.render_to_response()` function, or something similar.

This has, in my opinion, a major drawback: it might limit code reuse and it increases the coupling in the code. Everything's not lost, however.

Before you start the flame wars, let me explain, using an example coming from the Django website[5]; this represents a basic Django view function, returning some response containing data fetched from the database:

```
from django.shortcuts import render_to_response, get_object_or_404
# ...
def detail(request, poll_id):
    p = get_object_or_404(Poll, pk=poll_id)
    return render_to_response('polls/detail.html', {'poll': p})
```

Let's say now that I want to reuse that particular data (the 'p' variable) in another view: given that the return value is always an HttpResponse instance, you are screwed; sometimes you just need the data, to find something, or simply to render it in another format like JSON or

[1] http://djangopeople.net/mbi/
[2] http://www.djangoproject.com/documentation/request_response/
[3] http://www.djangoproject.com/documentation/request_response/
[4] http://classics.mit.edu/Hippocrates/hippolaw.html
[5] http://www.djangoproject.com/documentation/tutorial03/

XML (RESTful architectures, anyone?). This goes pretty much against the DRY[6] principles, and if you don't go deeper than the Django tutorials, your whole application might feature lots of repeated code.

Even worse, you have a direct reference to a template ("polls/detail.html"), and this kind of coupling does not scale well. It can become a real problem in big projects.

There are, however, strategies to avoid this: the first, the most common, is to refactor your code and to create a "layer" of data-specific functions, which will return instances (or arrays thereof) that you can reuse here and there. Doing this in a big project already started requires a good deal of unit testing first, to ensure that your refactoring is not breaking something elsewhere, but that's another problem (because you DO unit test, right??). This approach might not scale well in complex projects, and thus you would like to organize your code in other ways.

I learnt about organizing views using callable objects[7] instead of functions while studying the code in the Django REST Interface project[8]. In this case, you create code like this[9]:

```
class Resource(ResourceBase):
    """
    Generic resource class that can be used for
    resources that are not based on Django models.
    """

# ... snip ...

def __call__(self, request, *args, **kwargs):
    """
    Redirects to one of the CRUD methods depending
    on the HTTP method of the request. Checks whether
    the requested method is allowed for this resource.
    """
    # Check permission
    if not self.authentication.is_authenticated(request):
        response = HttpResponse(_('Authorization Required'), mimetype=self.mime
        challenge_headers = self.authentication.challenge_headers()
        response._headers.update(challenge_headers)
        response.status_code = 401
        return response

    try:
        return self.dispatch(request, self, *args, **kwargs)
```

[6] http://en.wikipedia.org/wiki/Don\XeTeXglyph\numexpr\XeTeXcharglyph"0027\relax{}t_repeat_yourself

[7] http://docs.python.org/ref/callable-types.html

[8] http://code.google.com/p/django-rest-interface/

[9] http://code.google.com/p/django-rest-interface/source/browse/trunk/django_restapi/resource.py

```
    except HttpMethodNotAllowed:
        response = HttpResponseNotAllowed(self.permitted_methods)
        response.mimetype = self.mimetype
        return response
```

The important bit here is the "**call**" method, which allows an instance to be called as a function, without specifying any particular method. This makes me remember of the dreadful VB default methods[10] but in Python it's not that bad, actually (VB is horrible by default[11] anyway), and allows you to use a cool syntax to do complex tricks ("command pattern" way of doing things, without the method call overload). And of course, since you are using an object-oriented approach, you can use polymorphism and inheritance to organize and reuse code as much as you can (or want).

Finally, Marco told me that his team uses another cool approach: they avoid returning HttpResponse instances from the views, and instead use Python decorators[12] to generate those. This way, you can achieve another neat separation of concerns, and you can reuse code simply and effectively.

I understand that the Python philosophy[13] cares about explicitness, but the "easy" way of processing requests in Django leads to trouble in big applications: increased coupling, reduced DRY, more headaches. I think you should use some code-reuse strategy in your Django code, but this, of course, is more an architectural problem than a Django problem.

[10] http://www.vbmigration.com/detknowledgebase.aspx?Id=309
[11] /blog/land-of-the-forbidden-maneuver/
[12] http://www.python.org/dev/peps/pep-0318/
[13] http://en.wikipedia.org/wiki/Python_philosophy#Programming_philosophy

wp-super-cache problem? Easy fix

Adrian Kosmaczewski

2008-04-23

I've just installed the excellent wp-super-cache plugin[1] to accelerate things a bit in this blog; today somebody sent one of my pages to reddit and I've had more users than usual! - by the way, thanks for coming! :)

Update: I admit, it also has a bit to do with the reading of today's entry in Coding Horror[2] ;) But I love WordPress nonetheless. I prefer it over MovableType (which I used during one year and a half).

The only glitch was: **at first, it didn't work**. I think you know the feeling.

The plugin was enabled, everything was activated, yet no files were cached: exactly this same problem[3]. And I found a quick fix to it, hence this post: fire your FTP client of choice[4] and go to /wp-content/cache. If you don't see a "supercache" folder in there, just create it. Magically, if you have some traffic on your blog and you dig in that folder a couple of seconds later you'll see already lots of files! No need to modify .htaccess whatsoever.

If you want to populate it, log out of the WordPress admin and start clicking on your blog. This plugin only creates cached files for anonymous visitors! I also turned on the compression so you should start having faster response times.

It was the only thing to do manually to get it working. Hope this helps! Any comments, as usual, more than welcome.

[1] http://wordpress.org/extend/plugins/wp-super-cache/
[2] http://www.codinghorror.com/blog/archives/001105.html
[3] http://wordpress.org/support/topic/161744?replies=25
[4] http://www.panic.com/transmit/

On the Need of Minimalist Polyglots

Adrian Kosmaczewski

2008-05-12

Many companies, at some point of their history, ask themselves a simple question: **what programming language should I use?** The answer to this question is tricky, and has big, big consequences, for every single line of code of your future products will be written, read and suffered by it. This single choice defines the level of salaries you will have to pay, the skills of programmers you will have to deal with, the relative length and performance of your systems, the availability of tools (or lack thereof), the kind of support you will get (or not), the number of operating systems your code will work in, etc.

Given the fact that Web Development equals Software Development[1], this discussion will be of interest to those building the smallest web-sites, as well as old desktop-intensive apps. It will not be a "Tell me what programming language you use, and I will tell you who you are" type of article, though it may look like one, because that is something you have to figure out all by yourself.

If you take a look at the list of programming languages[2], any business person would have an instant headache. There are lots of them. With the strangest names. You cannot possibly guess which one to pick from such a list, obviously. So, how do companies choose the languages they use? There are some straightforward methods that I have seen so far, in no particular order:

- Looking at what other companies use (typically Google[3], Microsoft[4], Apple[5] or Sun[6], but it could be 37signals[7] too).
- Following the advice of the CIO, the Lead Architect or some other politically-powered person, which might or might not have read this article ;)
- Looking at what the current pool of programmers in the company know how to use. Rinse, wash, repeat.
- Following hype.

[1] /blog/web-development-is-software-development/
[2] http://en.wikipedia.org/wiki/List_of_programming_languages
[3] http://www.google.com/
[4] http://www.microsoft.com/
[5] http://www.apple.com/
[6] http://www.sun.com/
[7] http://37signals.com/

- Because there is a market plenty of available, cheap programmers that I could use for this project.
- Taking into account the characteristics of the languages themselves (static vs. dynamic, etc).
- Following the company's history of past projects (successful or not).
- Following what your the client suggests (or mandates).

I think that **it is a very bad idea to take any of the above methods in isolation, without considering other factors.** Doing so is a path to self-destruction in the medium to long term, even if you succeed in the short term.

Just as a small background for people not into programming: you can safely (roughly) group programming languages in a table like this:

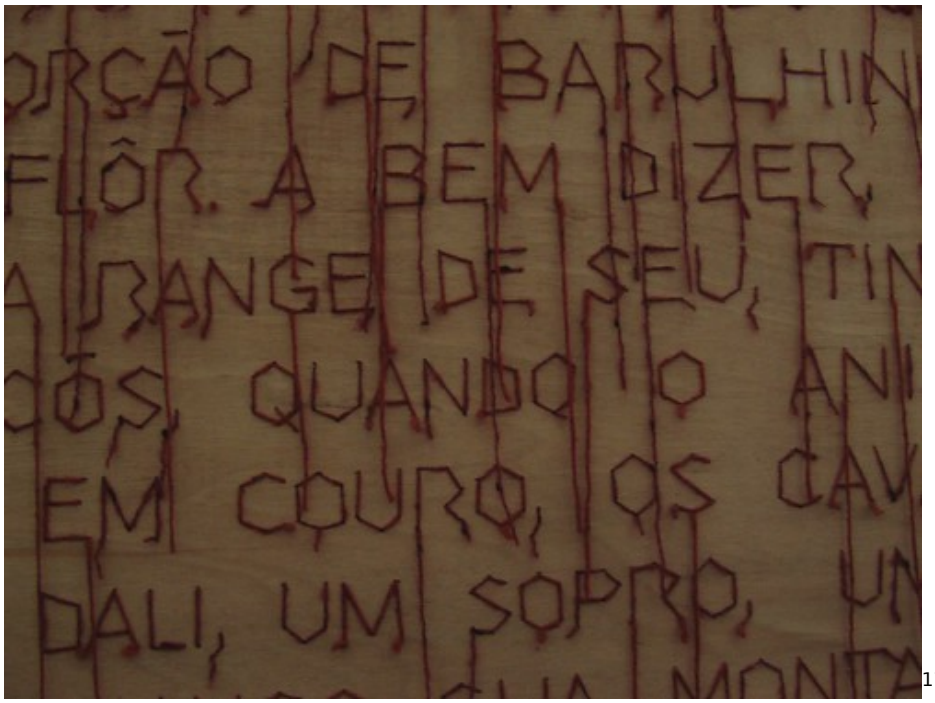
	Static	Dynamic
Strongly typed	Java[8], Objective-C[9], Pascal, ...	Python[10], Ruby[11], ~~JavaScript~~, Objective-C, Lisp, ...
Weakly typed	C++[12], C, ...	JavaScript[13], VBScript[14], ...

In a static language all variable type references are bound at compile time; in a dynamic language, this is done at runtime (which allows you to assign a string or an int to the same variable). In a strongly-typed language, either the compiler or the runtime enforces the operations you can and cannot do on an object (depending on its type, as you may have guessed). In a weakly-typed one, there is no such restriction, and you can perform implicit conversions from one type to the other. And then you have functional ones, but that is another problem, because there are many programming paradigms[15] out there. And then there is the hybrid ones, which I love as Steve Yegge[16] does:

> But also there's, like, the Boo language, the io language, there's the Scala language, you know, I mean there's Nice, and Pizza, have you guys heard about these ones? I mean there's a bunch of good languages out there, right? Some of them are really good dynamically typed languages. Some of them are, you know, strongly [statically] typed. And some are hybrids, which I personally really like.

[8] /blog/not-exactly-what-i-meant/
[9] /blog/iphone-sdk-une-nouvelle-ere-démarre/
[10] /blog/my-first-django-project/
[11] /blog/deliver-now/
[12] /blog/templates/
[13] /blog/javascript-tips-tricks-1/
[14] /blog/land-of-the-forbidden-maneuver/
[15] /blog/about-oop-and-other-programming-paradigms/
[16] http://steve-yegge.blogspot.com/2008/05/dynamic-languages-strike-back.html

He did not include Objective-C as “hybrid”, but I think it is. Anyway, so much for the theory, here is the main point of this article:



[17]

First of all, I consider programming languages (I know a few of them[18], and I have my personal picks[19]) just as tools to get things done[20] ™. **Nothing else.** I think of them as hammers or Black & Decker screwdrivers or saws or nail guns.

Second, I believe that specialization is for insects[21]. Getting stuck in a single programming language because of any of the reasons I have enumerated above is just stupid.

So what I want to say is: **You need polyglot programmers in your team**, like you need a team of people knowledgeable in many human languages in every company doing business at global scale. I would say even more, you need not only people fluent in western languages (like English, French, Spanish and Italian, which is my combination) but also in other languages, with different paradigms behind, like Arab, Hebrew, Hindi or Chinese.

What does that mean in programming terms? You want to have programmers in your team being able to use different languages in different ways; you want to have programmers that learn new languages every so often, just for the sake of it. And most importantly, you

[17] https://www.flickr.com/photos/urban_data/373580375/
[18] /blog/erlang/
[19] /blog/preferred-programming-languages/
[20] http://www.davidco.com/what_is_gtd.php
[21] http://www.elise.com/quotes/a/heinlein_specialization_is_for_insects.php

want to get rid of programmers that not only get stuck on a single language, but that, even worse, dismiss any other way to do things. Having people that takes a negative look on the learning side of things can bring your whole company to a dead-end. This industry is plenty of integrists, and you do not want that in your company.

For example, JavaScript can be used as a procedural, object-oriented or functional programming language. Does your JavaScripters know how to write functional JavaScript? If not, that is too bad, because they will not be able to fully understand what is going on behind the scenes in Prototype[22] or jQuery[23] then. Of course this will not block them from using those libraries, but they might not understand how to apply some interesting patterns in their own code.

Ask more about your programmers: do they know how to do complex C++ template metaprogramming[24]? Are they aware of some performance problems brought by garbage collectors[25]? Do they follow the latest evolutions of the next version of JavaScript? (even if they do not like them[26]) Which blogs[27] do they follow? Do they know why some people hate PHP[28]? Do they know what a continuation server[29] is? Which programming books have they read[30] lately?

And finally, what is even more important than having chosen a good programming language? **Your methodology.** Ask yourself (or your team) about these points:

- Do you write unit tests? You might not have testers, which is a bad idea[31] anyway, but that does not block you from testing code yourself (Steve Yegge[32]):

  > And I would say it's a pain in the butt, but I mean... it's a pain in the butt because... a static type-systems researcher will tell you that unit tests are a poor man's type system. The compiler ought to be able to predict these errors and tell you the errors, way in advance of you ever running the program.

- Do you use defensive programming[33] techniques?
- Do you have a wiki, a project website or at least good documentation in your code? If not, how do your new hires learn about your product? No, reading the code is NOT a good answer.

[22] http://prototypejs.org/
[23] http://jquery.com/
[24] http://en.wikipedia.org/wiki/Curiously_Recurring_Template_Pattern
[25] /blog/common-garbage-collection-problems/
[26] /blog/now-this-is-ridiculous/
[27] /blog/6-blogs/
[28] http://www.ihatephp.net/
[29] http://www.seaside.st/
[30] /blog/best-books-of-2007/
[31] http://www.joelonsoftware.com/articles/fog0000000067.html
[32] http://steve-yegge.blogspot.com/2008/05/dynamic-languages-strike-back.html
[33] /blog/design-by-contract/

- Is your chosen programming language portable? You might think that your system will never have to run on Linux or Mac, but why stuck yourself on purpose?

The common factor of all the above items is that you have to **manage complexity**. If you write code, you are creating complex stuff, and one of the best ways to manage complexity is to create small systems; as Steve Yegge said[34]:

> Small systems are not only easier to optimize, they're possible to optimize. And I mean globally optimize.

 [35]

And this is why you do not only need polyglot, but also **minimalist programmers** in your team. Small is beautiful. Paul Graham (of Y Combinator[36] fame) knows that dynamic languages yield small systems[37]:

> The right tools can help us avoid this danger. A good programming language should, like oil paint, make it easy to change your mind. Dynamic typing is a win here because you don't have to commit to specific data representations up front. But the key to flexibility, I think, is to make the language very abstract. The easiest program to change is one that's very short.

[34] http://steve-yegge.blogspot.com/2008/05/dynamic-languages-strike-back.html
[35] https://www.flickr.com/photos/simon_shek/80220468/
[36] http://ycombinator.com/
[37] http://www.paulgraham.com/hp.html

And how can you get small programs? By choosing the right programming language. Which brings us to the beginning of this post! So, here go some tips for all of you looking for the right programming language:

- **Prefer strongly-typed, dynamic languages;** there are a lot of reasons for that, particularly those exposed by Steve Yegge in his excellent presentation at Stanford[38]:

  > Yeah, sure, it catches a few trivial errors, but what happens is, when you go from Java to JavaScript or Python, you switch into a different mode of programming, where you look a lot more carefully at your code. And I would argue that a compiler can actually get you into a mode where you just submit this batch job to your compiler, and it comes back and says "Oh, no, you forgot a semicolon", and you're like, "Yeah, yeah, yeah." And you're not even really thinking about it anymore.
  >
  > Which, unfortunately, means you're not thinking very carefully about the algorithms either. I would argue that you actually craft better code as a dynamic language programmer in part because you're forced to. But it winds up being a good thing.

Some points about his talk:

- Many of the caveats of dynamic languages are not (so) true anymore (like the lack of decent IDEs[39] or the performance problems);
- There is a lot of research going on nowadays on the performance of programming languages, and this means that code written in these languages will benefit from many improvements in the near future, for free;
- And yes, there is the hype factor I have mentioned above; the cool kids are using them[40]:

  > Which makes them exactly the kind of programmers companies should want to hire. Hence what, for lack of a better name, I'll call the Python paradox: if a company chooses to write its software in a comparatively esoteric language, they'll be able to hire better programmers, because they'll attract only those who cared enough to learn it. And for programmers the paradox is even more pronounced: the language to learn, if you want to get a good job, is a language that people don't learn merely to get a job.

- **Teach yourselves;** setup internal workshops to have all your team learn how to do cool stuff with those languages you have

[38] http://steve-yegge.blogspot.com/2008/05/dynamic-languages-strike-back.html
[39] http://www.jetbrains.com/idea/
[40] http://www.paulgraham.com/pypar.html

read about in DDJ[41].

- **Teach the community around you, and do not be scared of competition;** have your team write papers, articles on business or technical journals, publish code as open source projects, and show that you can go beyond.
- **Keep an open mind:** continuation servers, Comet[42] or multi-core processors[43] are the future. Is your team prepared?

Software is a social process. Once you get this in your mind, and get your team working proactively, collaboratively and teaching each other, the choice of a programming language comes in second place.

[41] http://www.ddj.com/

[42] http://en.wikipedia.org/wiki/Comet_(programming)

[43] http://www.gotw.ca/publications/concurrency-ddj.htm

Mention on TUAW

Adrian Kosmaczewski

2008-05-15

Article published on The Unofficial Apple Weblog[1].

> Okay I promise this is that last time I'm talking about the WWDC schedule (I can't speak for my confreres). Earlier today Brett posted on a nice Ruby script to convert the data to a readable PDF. Yesterday when we first noted that the WWDC schedule was available I complained about the lack of iCal compatible files and apparently Adrian Kosmaczewski agreed. He proceeded to cook up a little script to make the necessary conversions. You can download iCal format ics files for the iPhone, Mac, and IT tracks at his website.

[1] http://www.tuaw.com/2008/05/16/wwdc-schedule-ajax-to-ical/

WWDC Schedules in iCal Format

Adrian Kosmaczewski

2008-05-19

For those attending WWDC, and wanting to have the module complete schedule in iCal (as suggested by TUAW[1]), I've created a small script[2] (using Prototype[3]) that will transform the JSON data[4] of the official WWDC schedule[5] into calendar files, which you can save and load into iCal.

This script was featured in another TUAW entry![6] Thanks to Mat[7] for the link!

Enjoy! I hope this will be useful to you. By the way, if you're attending WWDC, just leave a comment below.

Update, 2008-05-26: Thanks to Jeff LaMarche[8] for posting a link to

[1] https://web.archive.org/web/20080526012319/http://www.tuaw.com/2008/05/15/wwdc-schedule-available/

[2] https://web.archive.org/web/20080522173428/http://kosmaczewski.net/wwdc2008/

[3] http://prototypejs.org/

[4] http://developer.apple.com/wwdc/data/grid.json

[5] https://web.archive.org/web/20080724150704/http://developer.apple.com/wwdc/schedules/

[6] https://web.archive.org/web/20080520180746/http://www.tuaw.com/2008/05/16/wwdc-schedule-ajax-to-ical/

[7] https://web.archive.org/web/20081029000645/http://www.tuaw.com/bloggers/mat-lu/

[8] https://web.archive.org/web/20091115094755/http://iphonedevelopment.blogspot.com/

the calendars from the cocoadev mailing list[9], and to Second Gear[10] for using the calendars with their own products[11]! :)

[9] https://web.archive.org/web/20081202235245/http://www.cocoabuilder.com/archive/message/cocoa/2008/5/23/207961

[10] https://web.archive.org/web/20080526152314/http://secondgearllc.com/

[11] https://web.archive.org/web/20080820014645/https://blog.secondgearllc.com/2008/05/25/use-today-to-keep-up-with-your-conference-schedule/

Pourquoi Pas?

Adrian Kosmaczewski

2008-05-21

Pourquoi ne peut-on pas avoir des conférences comme celle-ci[1], avec des mini-events comme celui-ci en parallèle[2] en Romandie? Ou encore comme celle-ci[3]? Ou bien comme cette autre[4]! Ou celle-là[5]!

Pourquoi pas? Est-ce que le marché est trop petit? Y a-t-il un manque d'intérêt général? Je pense que le problème est important, et qu'une grande partie de l'élan technologique local est étouffé par ce manque d'une grande conférence technique chez nous.

Après tout, le web a été inventé à Meyrin[6].

Je constate avec amertume que la Suisse est à la traí®ne dans ce domaine. En Europe il faut se déplacer vers Londres (parfois Paris ou Barcelone, parfois Prague ou Copenhague) pour pouvoir assister à des événements techniques du type "world-class", avec des speakers qui font vraiment la différence à niveau mondial. Supposez pendant un instant que Reg Braithwaite[7], Adrian Holovaty[8], Leah Culver[9], David Heinemeier Hansson[10], John Resig[11], Tim O'Reilly[12], Christophe Porteneuve[13], Steve Yegge[14], Andy Clarke[15] et Avi Bryant[16] venaient à Palexpo[17] pour une série de conférences et d'expositions: n'iriez-vous pas les écouter? (Vous ne connaissez pas tous ces noms? Je vous invite a suivre les liens et a voir ce qu'ils ont à dire et voir ce qu'ils font. Vous me comprendrez alors.)

[1] http://www.meshconference.com/
[2] http://www.meshconference.com/meshu/
[3] http://futureofwebapps.com/
[4] http://www.web2con.com/
[5] http://www.futureofwebdesign.com/
[6] http://www.cern.ch/
[7] http://weblog.raganwald.com/
[8] http://www.holovaty.com/
[9] http://leahculver.com/
[10] http://www.loudthinking.com/
[11] http://ejohn.org/
[12] http://tim.oreilly.com/
[13] http://www.tddsworld.com/en/
[14] http://steve-yegge.blogspot.com/
[15] http://www.stuffandnonsense.co.uk/
[16] http://blog.dabbledb.com/
[17] http://www.geneva-palexpo.ch/

Et pourtant! Des sociétés internationales, ce n'est pas ce qui manque. Des salles de conférence? Il y en a aussi, surtout à Genève. Chaque année les "TechDays" de Microsoft[18] font le plein. J'ai vu Apple remplir une salle de conférence aux Pâquis. Des sociétés spécialisées dans le web (ou le développement logiciel en général)? Il y en a des centaines, avec des noms comme Google et Yahoo! présents dans les environs. Des écoles spécialisées? Il n'en manque pas.

Conclusion: le marché n'est pas trop petit, et il n'y a pas non plus un manque d'intérêt.

Alors?

Pour tout vous dire, je me rappelle de Telecom 95[19] avec une certaine nostalgie. (Du coup, je me souviens de me connecter, depuis les terminaux dans les couloirs de Telecom 95, via Telnet vers les ordis de l'Uni de Genève pour vérifier mon e-mail, ou faire un "talk" avec un ami qui était du côté du Quai Ansermet, et de me retourner et de voir derrière moi une bonne dizaine de personnes, me regardant comme si je faisais partie d'un film de science fiction. Il était 1995, donc.)

Au delà de ces souvenirs, Telecom était une vraie conférence technologique, de pointe, aux portes de la Suisse, avec les grandes et petites sociétés à portée de main. Mais les organisateurs ont senti que ce n'était pas ici que les choses intéressantes se passaient, laissant la Suisse sans une énorme vitrine dans le monde. Depuis, tout ce qui concerne le software, le web, le design, l'innovation, à l'air de se passer **ailleurs**.

Certes, il y a Lift[20], mais qui est plutôt centrée sur la connexion entre technologie et société. Comprenez-moi bien: Lift est formidable; mais je rêve d'une conférence sur la technique, sur les tendances, sur le web, où l'on explique "live" le pourquoi du succès de certaines sociétés, où les laboratoires montrent leurs nouveaux langages de programmation, où les blogs se remplissent de nouveautés, où Twitter explose 1000 fois par jour, où l'on apprend que certaines idées reçues à propos du métier d'informaticien sont carrément fausses, et qu'on peut faire mieux.

En voilà un rêve.

[18] http://www.microsoft.com/switzerland/msdn/fr/techdays/default.aspx
[19] http://www.itu.int/TELECOM/wt95/index.html
[20] http://liftconference.com/

Felicidades De Inmigrante

Adrian Kosmaczewski

2008-05-22

Este es un mensaje para SpinDoctor[1]: en Ginebra se consigue Mantecol[2]!!! He aquí la prueba (la latita de Aromat[3] es para probar que la foto la saqué en Suiza ;)

[1] http://portenosenginebra.blogspot.com/

[2] http://www.youtube.com/watch?v=85SgBT0OeBw

[3] http://www.amazon.com/Knorr-Purpose-Seasoning-3-Ounce-Canister/dp/B000F3S6JY

En la foto: yerba mate Rosamonte[4], Dulce de Batata Canale[5] (con chocolate!), Mantecol y Alfajores Havanna[6]! Donde se consiguen? Como siempre, en la Bodega Latina de la Rue Caroline en las Acacias[7], en Ginebra!

[4] http://www.rosamonte.com.ar/home.php

[5] http://www.labuenavida.cl/content/view/33706/Dulce_de_Batata_y_Queso.html

[6] http://www.havanna.com.ar/

[7] http://portenosenginebra.blogspot.com/2007/11/comprar-yerba-ginebra-suiza.html

Sin embargo, atención! También están vendiendo un supuesto Dulce de Leche “Chimbote” (con la marca que muestro aquí a la izquierda[8], distribuido por esta empresa “Franco-Argentine”[9]) que personalmente creo que es un burdo plagio, una copia, una aberración. No es el original. No sé como explicarlo, pero para mi que no es. No es el mismo gusto, ni el mismo logo, claro esta. Ojo, no esta feo, pero no es el verdadero Chimbote, que viene en la lata que ven aquí a la derecha.

[8] http://www.francoargentine.com/Espanol/chimb_esp.htm
[9] http://www.francoargentine.com/

No se dejen engañar! :) Mira lo que te digo: el frasco de dulce de leche que vende la empresa “Franco-Argentine” no tiene **en ningun lugar** (ni la etiqueta, ni la tapa, ni nada) la indicación “Industria Argentina”. En ningún lugar! Este producto no es argentino, no está hecho en Argentina, es cualquiera. Es falso. No es el Chimbote famoso y delicioso. Es una copia. He dicho. Indignado el hombre.

Aca encontre[10] otro “Chimbote”, que obviamente, nada que ver. Una vergüenza.

[10] http://www.deleite.es/productos/dulces.asp

Mention on Second Gear Software's Blog

Adrian Kosmaczewski

2008-05-24

Published on Second Gear Software blog[1].

> In a few weeks, the most hardcore of the Apple faithful, the developers, will be descending upon San Francisco for this year's Worldwide Developers Conference. Apple recently released the full schedule of sessions that will be taking place during the session. Using a mix of Today and Adrian Kosmaczewski's WWDC iCal creator you can have a quick way to see what sessions you plan to be at each day.

[1] http://blog.secondgearsoftware.com/2008/05/use-today-to-keep-up-with-your.html

Imagemagick

Adrian Kosmaczewski

2008-05-28

ImageMagick[1] is a cool toolkit; not only it's a complete set of command-line applications, ported to Windows, Mac and Linux, supporting hundreds of different image formats[2], it's also a C++ library[3] that you can use in your own applications!

On Mac OS X, I installed it via MacPorts[4] using the all-time classic:

```
sudo port install ImageMagick
```

Then I created a C++ command-line application with Xcode, set the header and library paths in the target properties (/opt/local/include/ImageMagick and /opt/local/lib in this case) and I was ready to code. The API documentation[5] and the tutorial[6] give some hints, and using those examples I've cooked a quick image transformation utility to play with:

```
#include <Magick++.h>

using Magick::Image;
using Magick::Geometry;
using Magick::Blob;

int main (int argc, char * const argv[])
{
    Blob blob;

    Image png;
    png.read("pic.png");
    png.write(&blob);

    Image jpg(blob);
    jpg.magick("jpg");
    jpg.zoom(Geometry(100, 200));
    jpg.write("newpic.jpg");
```

[1] http://www.imagemagick.org/
[2] http://www.imagemagick.org/script/formats.php
[3] http://www.imagemagick.org/Magick++/
[4] http://www.macports.org/
[5] http://www.imagemagick.org/Magick++/Documentation.html
[6] http://www.imagemagick.org/Magick++/tutorial/Magick++_tutorial.pdf

```
    Image tiff(blob);
    tiff.magick("tiff");
    tiff.zoom(Geometry(3000, 84000));
    tiff.write("newpic.tiff");

    Image gif(blob);
    png.magick("pdf");
    png.write("newpic.pdf");

    return 0;
}
```

Interesting stuff indeed! The API provides a complete set of "Photoshop-like" operations on images, which I plan to study further in the near future.

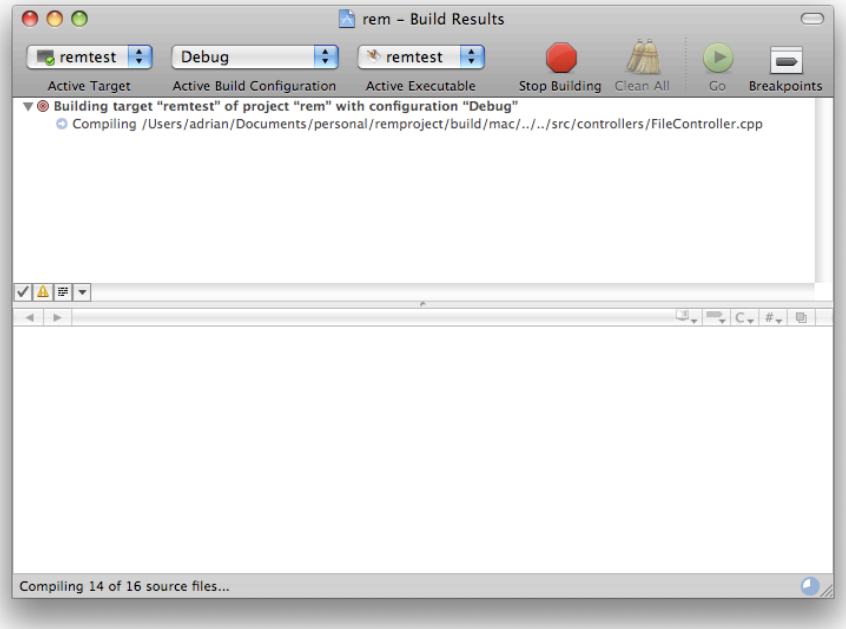
Amazing Xcode

Adrian Kosmaczewski

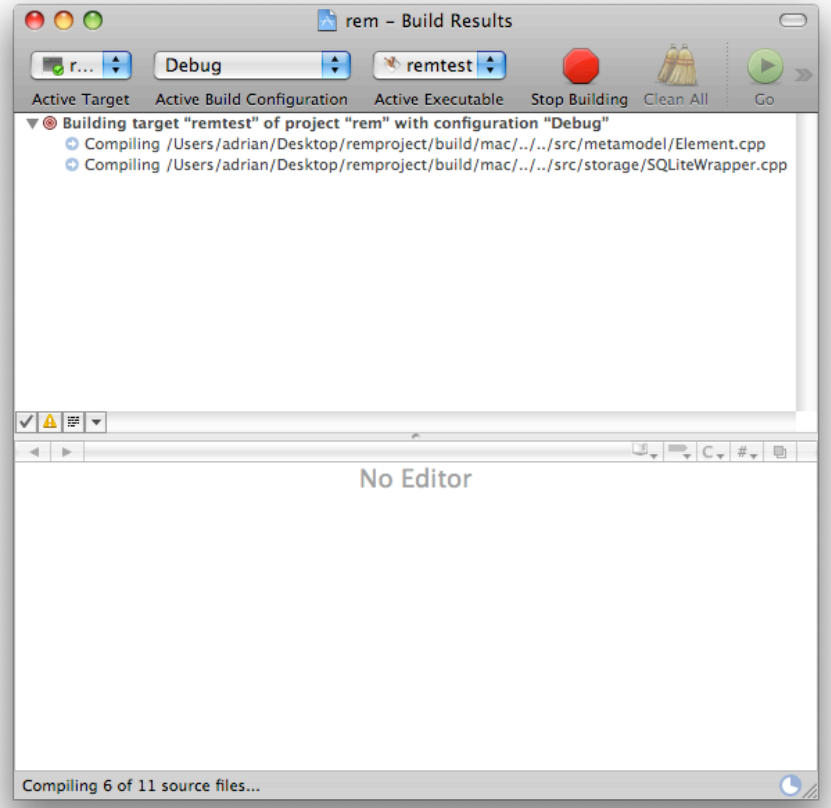
2008-05-29

Xcode is amazing. Of all the IDEs I've used (and this is, as always, a personal opinion, having used Visual Studio since version 6, Eclipse, Kdevelop and others) it's the one I prefer. And today I found another reason to like it.

I've noticed this: when I use Xcode on a single-processor machine (such as a Powerbook G4) and I rebuild my master thesis project (in C++) I see this build window:



No big deal: each file is compiled, one after the other. However, it seems that when I run it on a dual-processor machine (both in my dual G5 desktop and my Intel Core Duo 2 MacBook) I get a boost in compiling time, with parallel compilations! See this:



Then, coupled with Bonjour networks, you can even go faster, using the processing time of your peers' computers to have smaller compilation times.

Interview sur IB com Magazine

Adrian Kosmaczewski

2008-05-31

Interview by Marie-José Jones, published in theJune 2008 issue[1] of IB com magazine[2] (download the PDF scan of the article[3]).

> IB com: Pour quelles raisons avez-vous passé de Ruby on Rails à Django? Adrian Kosmaczewski: Chez Electronlibre, nous pensons que chaque framework à un “target” spécifique, et que chaque projet doit bénéficier d’une analyse particulière pour déterminer la technologie la plus adaptée.

[1] http://www.ibcom.ch/ee01/news/2008/06/

[2] http://www.ibcom.ch/

[3] /press/IBCOM_June_2008.pdf

Sunny WWDC

Adrian Kosmaczewski

2008-06-07

Comparing Lausanne to San Francisco is a straightforward experiment:

By the way, I’m in SF. On Monday I’ll attend WWDC. But tomorrow Claudia and I are biking through the Golden Gate ;)

Pastrami Sandwich

Adrian Kosmaczewski

2008-06-10

I find many similarities between an event like WWDC[1] and a similar one I've attended at Redmond long ago[2]; both are big (huge!) events, with thousands of (men) engineers from around the world (and very few women), with a keynote by the founder[3], lots of events every morning and afternoon, and merchandasing stuff all over the way. And of course, in both cases you get food boxes for lunch[4].

However, there is one basic difference between both events. Apple not only has interesting technologies to show up, even bleeding edge ones, more often than not on the open[5] and public[6] domain[7] (many of which I can not write about, and boy they are going to make a difference!), but even better than that, **it has a vision.**

And passion. Cocoa developers are among the most passionate I've ever met, and you just can't find that in a Microsoft event. You can feel that in the (conditioned) air of the Moscone center, almost touch it. New projects everywhere. People discussing about their ideas. Lots of collaboration, openness and willingness to go further. Microsoft's stuff is, well, boring at best; dull and gray. Enterprise IT is no fun, believe me, but there's no reason to try to look at it in a different way. And faithful to its own way, Apple is precisely doing that, right now; and what's about to come will reshape the industry forever.

As Steve Jobs announced yesterday[8], the iPhone is now ready for the enterprise, given that the majority of companies here and there use Exchange as the basic means of communication. Now the iPhone is ready to access all that information, and I while I can't tell you more, this is the just the first step. iPhones will be an important player in the enterprise landscape in the future months.

On the other hand, well, Microsoft is asleep. The sales of Macs are booming, and more and more shops are shifting to it. The iPhone is

[1] http://developer.apple.com/wwdc/
[2] /blog/bill-gates-at-the-mosdc-2006-live-blogging/
[3] http://www.apple.com/quicktime/qtv/wwdc08/
[4] /blog/microsoft-losing-its-focus/
[5] http://www.zeroconf.org/
[6] http://webkit.org/
[7] http://llvm.org/
[8] http://www.apple.com/quicktime/qtv/wwdc08/

sold out, the WWDC is sold out, a new wave of iPhones is about to hit the street, and there's much, much more in store for the future.

Personally, I think that the center of gravity of the consumer software industry has shifted (pay attention to the tense of the verb I used). There is one and only clear leader, and that's Apple. Microsoft should concentrate its effors in what it does best (I think), which is enterprise products, and stop delivering crappy operating systems every 5 years. Any advantage they had 10 years ago has just vanished, and that's mainly Microsoft's own fault. Ballmer should have resigned long ago[9]. But again, as Joel said[10], they have enough cash to continue screwing things up for a long time.

In the meantime, I'm finishing my pastrami sandwich, sitting right beside the Mission conference room, waiting for the next session. I'm actually here in San Francisco and I can barely believe it.

[9] /blog/yeah-i-work-using-technology-made-by-this-company/
[10] http://www.joelonsoftware.com/articles/APIWar.html

Basic vs Digest

Adrian Kosmaczewski

2008-07-07

In the series of highly boring posts ;) here's another one; in this case, a simple explanation of two different authentication protocols available in the HTTP standard.

HTTP Basic Authentication Protocol

This is the simplest HTTP Authentication protocol available:

- The browser sends a request to a protected resource: `GET /index.html`
- The server looks for the "Authenticated" header in the request; since it does not find it, it replies back with a response with the 401 code ("Unauthorized"). The response contains a "WWW-Authenticate" header, with the value "Basic". This response is called a "challenge", and it also contains a "realm" text, which describes the protected resource in clear text (the "realm" is shown in the pop-up window that usually appears for you to type your password when this protocol is used).
- The browser creates a new request `GET /index.html` that contains an HTTP_AUTHORIZATION header, whose value is the word "Basic" followed by the 'username:password' string encoded in base 64. This algorithm is a two-way algorithm: you can retrieve the username and password from the base 64-encoded string.
- The server receives this response, and since base 64 is a two-way algorithm, it compares the MD5 (or SHA1) password hash to the one stored in the database. If they are the same, the request is processed. Otherwise, the user gets a 401 again.

The advantage of this protocol is that it is simpler to implement, but the tradeoff is that any malicious user sniffing on the network can retrieve the username:password combo and use the base 64 algorithm to get the original values. All of this makes this protocol relatively insecure.

HTTP Digest Authentication Protocol

Simply put, the HTTP Digest Authentication protocol goes like this:

- The browser sends a request to a protected resource: `GET /index.html`
- The server looks for the “Authenticated” header in the request; since it does not find it, it replies back with a response with the 401 code (“Unauthorized”). The response contains a “WWW-Authenticate” header, with the value “Digest” followed by two long MD5 strings (“nonce” and “opaque”) generated specifically for this request, and that the browser will use to answer back to the server. This response is called a “challenge”, and it also contains a “realm” text, which describes the protected resource in clear text (the “realm” is shown in the pop-up window that usually appears for you to type your password when this protocol is used).
- The browser creates a new request GET /index.html but this time it will not send the username/password combo in clear text (like in the case of HTTP Basic Authentication): it will “hash” this information, together with the “nonce” and “opaque” values received from the server, together with the “realm”, using the MD5 hashing algorithm. This is a username/realm/password hash, so to speak.
- The server receives this response, and since MD5 is a one-way algorithm (you cannot, theoretically, find the original password from all the blah-blah sent in step 3), it compares the username/realm/password hash to the one stored in the database. If they are the same, the request is processed. Otherwise, the user gets a 401 again.

The advantage of the HTTP Digest Authentication protocol is that the key exchange between client and server is done in encrypted hashes. Even better, the server does not store the password per se, but a complex mix of strings all hashed together. This makes the protocol a very secure option, but at the same time, it requires more processing time and code to execute.

However, the HTTP Digest Authentication protocol implies several design issues:

- The username/realm/password combo is encoded and stored using the “realm” string. So this means that if the realm changes, all the passwords become invalid. And since MD5 is a “one-way” algorithm, you cannot retrieve the passwords, and your users must retype them. Be careful!
- Since the default user classes in most web programming toolkits (Django, Rails, etc) just store an MD5 (or SHA1) hash of the password, to use HTTP Digest Authentication you must have a separate database field, somewhere, to store the username/realm/password combo hash.

The Beauty of Cocoa

Adrian Kosmaczewski

2008-07-08

(Highly geeky post ahead. You've been warned!)

I have found the very message that summarizes the beauty of Cocoa in a single word; see by yourselves, hereunder in line 47:

```
#import <Foundation/Foundation.h>

// The interface of a person
@interface Person : NSObject {
    NSString* firstName;
    NSString* lastName;
    int age;
}
@end

// The implementation of the Person
@implementation Person
-(id)init {
    if (self = [super init]) {
        firstName = @"";
        lastName = @"";
        age = 0;
    }
    return self;
}

-(void)dealloc {
    [firstName release];
    [lastName release];
    [super dealloc];
}

-(NSString*)description {
    return [[NSString alloc]
            initWithFormat:@"Name: %@ %@, %d years old",
            firstName, lastName, age];
}
@end
```

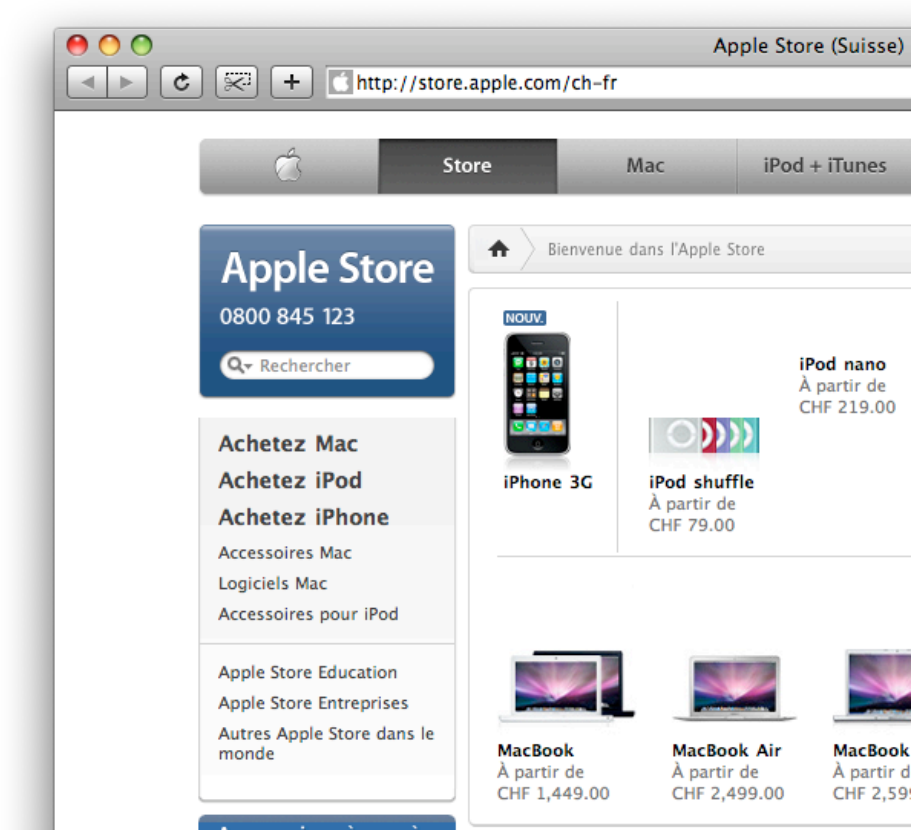
```objc
// Some code using the Person class
int main (int argc, const char * argv[]) {
    NSAutoreleasePool * pool = [[NSAutoreleasePool alloc] init];

    NSMutableDictionary* dict = [[[NSMutableDictionary alloc] init] autorelease
    [dict setObject:@"Teto" forKey:@"firstName"];
    [dict setObject:@"Rodriguez" forKey:@"lastName"];
    [dict setObject:[[NSNumber alloc] initWithInt:34] forKey:@"age"];

    Person* person = [[[Person alloc] init] autorelease];

    // The beauty of Cocoa can be resumed to this very line:
    [person setValuesForKeysWithDictionary:dict];

    // Now sit back and relax:
    NSLog([person description]);

    [pool drain];
    return 0;
}
```

iPhone at the Swiss Apple Store

Adrian Kosmaczewski

2008-07-10

Available right now:



A Watch - from an OOP Perspective

Adrian Kosmaczewski

2008-07-13

A watch might be one of the most common types of objects, but it remains also one of the earliest pieces of human craftmanship to show an extreme level of complexity, all contained in a small amount of space. Since the late 1700s, artisan watchmakers in Switzerland and elsewhere have shown their pride and skills creating watches called "Grande Complications", containing thousands of individual parts and performing incredible functions:

> The most complicated watch ever made, known in watch enthusiasts' circles as "The Ultimate Watch," is Patek Philippe's "Calibre 89.
>
> The incredibly precise operation of 1728 parts in this really ultimate masterpiece of watchmaking allows to perform no less than 33 (thirty-three!) complicated functions, among them a correction for the 400-year-rule, an Easter date indication, a star chart, a tourbillon, a perpetual calendar, a sidereal time indication, and, and, and ...
>
> This watch was sold in 1989 for the nice round sum of about four million Swiss francs.

(Ozdoba, 2005)

More information about the "Calibre 89" can be found here[1] and in the Patek Philippe Museum website.

However, the same watchmakers that made these fine pieces were also aware of the basic information that their creations were to provide: **time**. As such, their watches remained extremely easy to use, and they set up the basic standard for analog watches, in such timeless designs that the latest Swatch models show the same basic layout and functionality.

The underlying concept is the very same used in today's object-oriented abstraction and encapsulation. Even Apple uses the idea of the watch to show this characteristic:

> All programming languages provide devices that help express abstractions. In essence, these devices are ways of

[1] http://marina.fortunecity.com/westindia/59/ppc89.htm

> grouping implementation details, hiding them, and giving them, at least to some extent, a common interface–much as a mechanical object separates its interface from its implementation.

In this article I will provide my view about how different OOP concepts apply to a real-life object such as a watch, in all its forms.

Concepts

Object

Every existing watch could be thought of a single instance of a more generic class “Watch”. This is possible since all watches share a common set of characteristics, that is, at least, the capacity to display the current time. Of course their external appearance and their whole set of characteristics may differ, but in this common aspect, they are all the same. This makes the “Watch” class a simple but extensible one.

Attributes

Every watch has a distinct set of attributes, for example:

- Brand
- Type of wristband (metallic, leather, plastic)
- Waterproofness
- Current time
- Size
- Cadran color
- Analogic or Digital
- Number of hands
- Battery level

Behavior

Watches have a basic behavior; they increment their “current time” attribute, by one second every second. This way, they manage to update themselves automatically, and to provide users with the right time all the time. Watches that provide different functionalities may also have different behaviors (I had long ago a Casio watch which provided barometric pressure and temperature, both constantly updated and very handy when doing outdoor activities).

Class

A “Watch” class would provide common characteristics to all watch instances, the most basic being that of providing time information and being able to be held by a person (his owner).

Inheritance

There are lots of different types of watches, but some common subsets can be easily seen:

- Swiss vs. Japanese
- Digital vs. Analog watches
- “Grande complication” vs. simpler watches
- Battery vs. wrist kynetics- powered
- Plastic vs. metal watches
- Quartz vs. mechanical

These subsets can be used to model the right class inheritance, the one that makes more sense in an application (since there is usually not a single answer). C++ provides also multiple inheritance, which is not the case of many languages; in this case, every instance could inherit from several base classes (Japanese + digital + simpler + battery + plastic + quartz = Casio watch).

At the same time, a “Watch” could be seen as a subclass of “Clock”, which also provides time information but is usually not easy to hold in the palm of your hand (the Big Ben is a clock, but not a watch).

Abstraction

Users are completely isolated from the internal mechanisms of their watch; they usually don’t know (nor they need to know) how their watches work; they just care about the current time displayed.

Modeling

When creating a new kind of watch, a designer might find inspiration on existing watches or from new functionalities that might be useful to the end user. This in turn “reshapes” an implicit Watch class, adding new subclasses, or properties to existing classes. This is something that Casio made extensively in the 80s and 90s, providing watches with extended functionalities, rather different from what was offered at the time (temperature, barometric pression, speed, even television or radio).

Messages

Whenever the user sets up the right time in a watch, or uses one of its functionalities (for example to get the current date, the temperature), she has to interact somehow with the watch; usually this is done using a set of controls (knobs or keys) located around or over the watch, and each time that the user presses or turns one of those controls, one could think of a message sent to the watch. Of course a correct protocol is needed (that is, turn or press twice to set the time to 2 o’clock) and some operations are forbidden, or even better, impossible (like setting the hour to 25, or the minutes to 345).

Encapsulation

This concept is closely related to that of “Abstraction”; users can use and interact with rather complex structures, using extremely simple commands, like buttons and knobs. This encapsulation guarantees a correct mechanism (it is difficult for users to screw up their watches), satisfies warranty requirements (since only qualified people can deal with the internal structure of the watches), and also simplifies the use of the watch, making it easy to use and providing value to the user.

Interface

Users do not need to know whether Patek Philippe watches are more complicated than Swatches, simply because their basic functionality is the same, and the difference has more to do with aesthetics (and price...) than anything else. Both provide the time in analogic form, with similar knobs and maybe even similar advanced functionalities (date, chronograph, etc).

Information hiding

The watch displays only the right amount of information to the user; the rest is kept internally, and is used by the internal mechanisms without the user even knowing about them.

Data members

Inside the mechanisms of the watch, implicit (in the case of analog watches) or explicit (in the case of digital watches) bits of information can be found; these could be either private (angles of the hands, resort tensions, controller values) or public (current time, battery level).

Member functions

Every distinct functionality or service of a watch could be thought of like a “member function” of the class Watch:

- Display current time
- Set time to new value
- Display current date
- Start chronograph
- Buzz alarm at the right time

Internally, the watch may have other “private” member functions, used to perform internal duties.

References

Apple Developer Connection, “Interface and Implementation”, [Internet] http://developer.apple.com/documentation/Cocoa/Conceptual/O

bjectiveC/Articles/chapter_3_section_2.html (Accessed October 1st, 2006)

CNN.com, “Patek Philippe – The ultimate watch”, Monday, October 31, 2005, [Internet] http://edition.cnn.com/2005/WORLD/europe/10/27/ultimate.watch/index.html (Accessed October 1st, 2006)

Ozdoba, Christoph, “Pocket Watches - Glossary”, August 13, 2005, [Internet] http://www.ozdoba.net/swisswatch/pocket_gloss.html#comp (Accessed October 1st, 2006)

Patek Philippe Museum, [Internet] http://www.patekmuseum.com/ (Accessed October 1st, 2006)

Swatch website, [Internet] http://swatch.com/index_flash.php (Accessed October 1st, 2006)

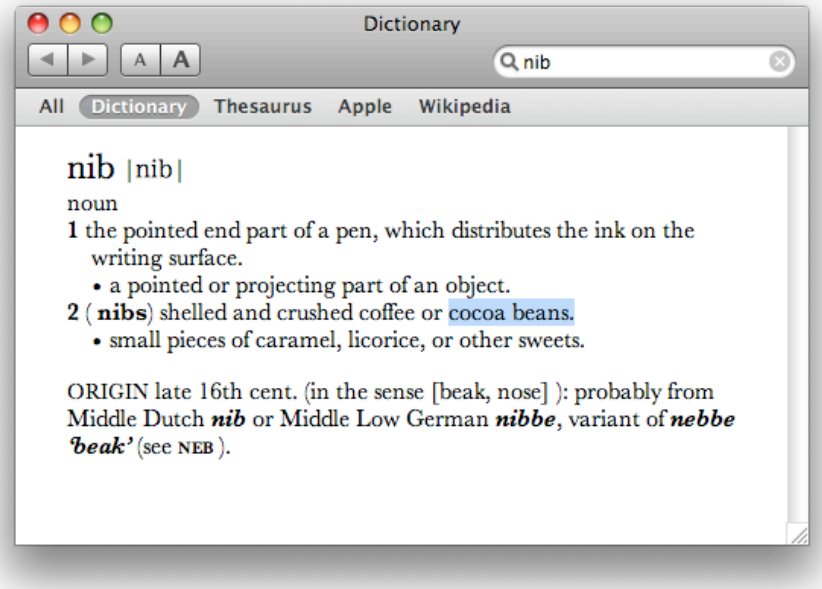
Webb, Ken, “Digital Watch - Sample Xholon App”, [Internet] http://www.primordion.com/Xholon/samples/watch.html (Accessed October 1st, 2006)

Nibs

Adrian Kosmaczewski

2008-07-14

I’m sure the pun between the acronym for “NeXT Interface Builder” and this definition of “cocoa beans” is intentional, but it surprised me anyway:



Challenges for Software Engineers

Adrian Kosmaczewski

2008-08-03

Software Engineering is the youngest of all the professions, being born around 50 years ago, but since then it has been continually improved. Practicers have fiercely debated upon it through the years, given the extremely fast pace of the innovations in the field, and the extremely difficult and inherently dynamic nature of software. Many trends have appeared and vanished, and many others will come.

In this article I will provide a short overview of two kinds of challenges that I consider that software engineers will have to confront in the next 20 years: the human and the technical.

The Human Factor

A quick look at the agenda of the 29th Int. Conference on Software Engineering (held in Minneapolis last year, from the 20th to the 26th May 2007) shows the key themes considered by the software engineering research community as the major challenges today:

- "Improving Software Practice through Education: Challenges and Future Trends"
- "Research Collaborations between Industry and Academia"
- "Model-driven Development of Complex Systems: A Research Roadmap"
- "Source Code Analysis: A Road Map"
- "Software Reliability Engineering: A Roadmap"
- "Global Software Engineering: The Future of Socio-technical Coordination"
- "Collaboration in Software Engineering: A Roadmap"
- "Self-Managed Systems: An Architectural Challenge"
- "Software Project Economics: A Road Map"

(Source: ICSE 2007)

Mixed up with technical concerns, some presentations highlighted core problems that appears in the current state of software engineering: **communication, collaboration and human issues.**

> The core substance of software deserves more eyes and more minds, thinking ways to describe not only the big picture (something that you can do with fancy diagrams) but

> also to give solutions to the problems that developers find daily while building systems up. Software is a process, but not any kind of process: a human one, maybe the most intangible of all processes; and as such, it is filled with all human brightnesses and failures.

(Myself, in 2006)

I have the deep, strong conviction that software development cannot and must not be separated from the human-side problems of forming, keeping and training teams, enhancing the internal and external communications, improving and enhancing the individual creativity as well as the ways of reaching team consensus. As a powerful example, the seminal Peopleware book by DeMarco and Lister showed that many of the most successful software companies have been those that excelled in creating human-centric environments:

> In 1982, (Mitchell Kapor) founded Lotus Development Corporation, for which he is most noted. While there, he revolutionized corporate workplace culture by making diversity and inclusivity top priorities in his goal for creating an environment that attracted and retained employees. There were many "firsts" for Lotus, including being the first company to sponsor an AIDS Walk event in the mid-80's and refusing to do business with South Africa due to Apartheid.

(Sterling-Hoffman)

Thanks to a sharp hiring process, a series of innovations in their flagship spreadsheet product, and a progressive corporate culture, Lotus dominated the software landscape of the 80s. Today, Google follows very closely Lotus' steps (Google, 2007a), and their brilliant results in the last few years seem to confirm this trend. Google for example allows their employees to use 20% of their time in their own projects (Google, 2007b). This is resulting in an incredible amount of code, used internally and also released as open-source projects:

> Google is a fantastic company to work for. I could cite numerous reasons why. Take the concept of "20 percent time." Google engineers are encouraged to spend 20 percent of their time pursuing projects they're passionate about. I started one such exciting project some time back, and I'm pleased to announce that Google is releasing the fruits of this project as an open source contribution to the Macintosh community. That project is MacFUSE, a Mac OS X version of the popular FUSE (File System in User Space) mechanism, which was created for Linux and subsequently ported to FreeBSD.

(Google Mac Blog, 2007)

The empowerment of both the individual **and** the team (the emphasis is important here) is key for a successful software project.

Parallelization

Herb Sutter has put it very clearly: technically speaking, since the beginning of the decade, there is no way for getting more processing power without jumping to multicore architectures:

> The key question is: When will it end? After all, Moore's Law predicts exponential growth, and clearly exponential growth can't continue forever before we reach hard physical limits; light isn't getting any faster.(…) If you're a software developer, chances are that you have already been riding the "free lunch" wave of desktop computer performance.(…) Right enough, in the past. But dead wrong for the foreseeable future.

(Sutter, 2005)

The problem is that **more cores do not necessarily mean more computing power**, because the jump done by chip manufacturers has not (yet) been completely followed by the software community. Of course there is the concept of "threads", and multi-threaded applications can benefit of performance boosts when running on multicore hardware platforms; however, a number of myths have to be debunked, as the common "2 x 3GHz = 6GHz" (as explained by Sutter here: http://www.ddj.com/showArticle.jhtml?documentID=ddj0503a&pgno=3), and even more importantly, creating multithreaded applications is not easy. At all.

A couple of months ago, in the "Questions and Answers" of LinkedIn.com I answered an interesting question about parallelization; the following excerpt of my answer pretty much summarizes my opinions about the current state of multithreading, as well as some challenges that are raised for the future:

> The problem is simply that the "streamline" programming languages languages do not provide good ways to code multithreaded applications. (…) Not at all. The problem is real, since multithreading applications are extremely complicated to think of, let alone develop properly. A line of code in a high-level language could mean several hundred instructions in a processor; and depending on the sharing algorithm used at the CPU level, each one of these instructions might be executed separately, sharing resources with other processes. So what happens when? (…)
>
> What I mean is that the fact that the JVM and the CLR support threads does not make good .NET or Java developers good multithreading developers by default. It's a different mindset; who is accessing your resources? (…)
>
> I think that as long as programming languages do not take multitasking and multithreading as base features (and not as mere library or API add-ons) we will continue struggling

> with single-threaded applications that collide with each other.

(Myself, this time on LinkedIn Answers, 2007)

I think that the challenge of parallelization is not only an extremely tough one, requiring what Thomas Kuhn calls a “paradigm shift”, but also an extremely huge business opportunity; after all, while the top of the Chinese ideogram for “Crisis” means “Danger”, the bottom part means “Opportunity” (Mary R. Bast, 1999).

Very Large Systems

I also think that software systems will invariably get bigger and bigger. And given the historically high risk of failure of software projects, the dependency on software of the modern society, the pervasiveness of the Internet, the low prices of connectivity and the overall globalization, it is more important than ever to get ready for those challenges.

In July 2006, the well known Software Engineering Institute of the Carnegie Mellon University published an impressive report (freely downloadable) called “Ultra-Large-Scale (ULS) Systems: The Software Challenge of the Future”:

> The study brought together experts in software and other fields to answer a question posed by the U.S. Army Office of the Assistant Secretary of the U.S. Army (Acquisition, Logistics & Technology): “Given the issues with today’s software engineering, how can we build the systems of the future that are likely to have billions of lines of code?” Increased code size brings with it increased scale in many dimensions, posing challenges that strain current software foundations. The report details a broad, multi-disciplinary research agenda for developing the ultra-large-scale systems of the future.

(SEI, CMU, 2006)

The 150-page long report gives an extremely detailed vision of the challenges raised by complex systems, in the following areas:

- Design
- Monitoring
- Human interaction
- Computational Engineering
- Deployment
- Legal issues

The report provides interesting conclusions, highlighting the methodologies and techniques that will required to tackle these systems efficiently, among them the role of the W3C, the forthcoming trends of grid computing and parallelization, the Model-Driven Architecture (MDA) initiative of the OMG, and finally the development of larger

Service-Oriented Architectures (SOA) platforms, such as .NET or J2EE (page 41 of the report).

The report also places a strong emphasis in the concept of socio-technical ecosystems and I think it's worth a read by everyone interested in software engineering.

Conclusion

Given its youth, we have yet to see the most important developments in software engineering. However, it is extremely difficult to predict the future in this industry: Bill Gates himself published a book in 1995, "The Road Ahead", where he only slightly talks about the World Wide Web:

> "The Road Ahead" appeared in December 1995, just as Gates was unveiling Microsoft's master plan to "embrace and extend" the Internet. Yet the book's first edition, with its clunky accompanying CD-ROM, mentioned the Web a mere seven times in nearly 300 pages. Though later editions tried to correct this gaffe, "The Road Ahead" remains a landmark of bad techno-punditry – and a time-capsule illustration of just how easily captains of industry can miss a tidal wave that's about to engulf them.

(Salon.com, 2000)

In any case, I think that there are important challenges in our industry: the need for better human management, the jump to multicore architectures and multiprocessing, and the ever-growing size of software projects. These three elements will without any doubt change the shape of the industry in the years to come, and raise new challenges in turn.

References

Adrian Kosmaczewski on LinkedIn Answers, "For the software architects out there, do you feel there is an impending paradigm shift in the software development model, towards"parallel computing" models?", January 2007, [Internet] http://www.linkedin.com/answers?viewQuestion=&questionID=7804&askerID=4194838 (Accessed June 3rd, 2007)

Adrian Kosmaczewski on Kosmaczewski.net, "What will the Software Architecture discipline look like in 10 years' time?", March 16th, 2006 [Internet] link[1] (Accessed June 3rd, 2007)

Bast, Mary R; "Crisis: Danger & Opportunity", 1999 [Internet], http://www.breakoutofthebox.com/crisis.htm (Accessed June 3rd, 2007)

[1] /blog/software-architecture-future/

DeMarco, Tom & Lister, Timothy, “Peopleware - Productive Projects and Teams, 2nd Edition”, 1999, Dorset House Publishing, ISBN 0-932633-43-9

Google, “Top 10 Reasons to Work at Google”, 2007a [Internet] http://www.google.com/jobs/reasons.html (Accessed June 3rd, 2007)

Google, “What’s it like to work in Engineering, Operations, & IT?”, 2007b, [Internet] http://www.google.com/support/jobs/bin/static.py?page=about.html (Accessed June 3rd, 2007)

Google Mac Blog, “Taming Mac OS X File Systems”, January 11th, 2007, [Internet] http://googlemac.blogspot.com/2007/01/taming-mac-os-x-file-systems.html (Accessed June 3rd, 2007)

ICSE, “Future of Software Engineering”, 2007, [Internet] http://web4.cs.ucl.ac.uk/icse07/index.php?id=104 (Accessed June 3rd, 2007)

Software Engineering Institute, Carnegie-Mellon University, “Ultra-Large-Scale (ULS) Systems - The Report”, July 2006, [Internet] http://www.sei.cmu.edu/uls/ (Accessed June 3rd, 2007)

Salon.com, 2000 [Internet], “Why Bill Gates still doesn’t get the Net”, [Internet] http://archive.salon.com/21st/books/1999/03/cov_30books.html (Accessed June 3rd, 2007)

Sutter, Herb; “A Fundamental Turn Toward Concurrency in Software”, 2005, [Internet] http://www.ddj.com/dept/architect/184405990 (Accessed June 3rd, 2007)

Sterling-Hoffman, “Opening Doors To Higher Education”, [Internet] http://www.sterlinghoffman.com/newsletter/articles/article140.html (Accessed June 3rd, 2007)

Wikipedia, “Thomas Kuhn” [Internet], http://en.wikipedia.org/wiki/Thomas_Kuhn (Accessed June 3rd, 2007)

Certification

Adrian Kosmaczewski

2008-08-05

While several other professions have a long, established and standard procedure of certification, the title “software engineer” is applied to both self-made developers, turned into experts of some technique, or to people with PhD degrees, and a long history of both academic and professional achievements.

When in some situations it is not legally possible to use the title “software engineer” without an engineering degree of some kind (for example, in some states of the USA or some institutions like the IEEE - http://www.ieeeusa.org/policy/positions/titleengineer.html), the term “software developer” is usually applied to people in charge of designing, writing and / or maintaining software-based systems. I will use the terms developer and engineer interchangeably in this discussion, which some people might think is not correct.

The discussion about the need of a formal certification process is a relatively new one:

> Professional certification in the IT industry is a relatively recent phenomenon. It was begun in the late 1980s by Novell, Inc., an upstart networking vendor from Provo, Utah, in an effort to build market share and manage support costs for its products by building the skill levels of the people who worked with those products. Novell was one of the first companies to recognize the links between education/skills and product success. They knew that they could not build an education infrastructure that would support their worldwide marketing plans with their own resources. However, they also recognized that if they did not provide for skills acquisition for their highly technical products, they could never meet their product revenue goals.

(Shore)

However, no consensus about whether or not certification is needed has been reached yet. This article will highlight some of the problems raised by software engineering certification, which might explain the lack of consensus cited before:

- The first one has to do with the inherent extension of the software engineering field: are all software developers equal?

- The second one has to do with the large number of available certifications: which one to choose? Which ones are “reliable” indicators of expertise, and in which fields?

What is a “Software Engineer”?

In my career, I’ve found self-made people (I’m one of them, actually), real-estate architects, lawyers, mathematicians, economists and even geophysicists writing code for a life. What I’ve seen so far is that the most successful software developers are those who like doing it, no matter which profession they’ve followed. And the opposite is also true: many guys with a computer science degree discover, some time after they start their careers, that they definitely do not like that code thing.

One of the biggest problems with certifications is that there is not such thing as a “single kind” of software developer:

- There are those who write games, and spend most of their time writing in low-level languages for game consoles, optimizing for speed and space, and creating three-dimensional worlds using as little memory as possible…
- There are those who write web-based applications, and spend their time creating 3-tier architectures, talking to a database, using some kind of object-oriented platform, and luckily exposing some data using XML web services, dealing with cross-browser issues, and wondering what is all that fuss about Web 2.0…
- There are those who write operating systems, and work for some embedded software company, or hack Linux kernel device drivers every night, or work for Microsoft or Sun or Apple, and spend most of their time discussing whether microkernels are better than monolithic architectures…
- There are those who have the ill fate of working as a consultant, and spend more time switching from project to project every day, or dealing more with corporate politics, rather than with code…
- There are those who manage projects and spend more time in their mailing list or in Microsoft Project rather than being able to code (and then complain about this in their blogs)…
- There are those who have a software engineering degree, but work for ZDNet writing about industry trends…
- There are those who turn into human resource consultants, and try to keep up to date on the new trends, but feel completely lost given what they learnt in university…
- There are those who do a little bit of all what I’ve mentioned above, and are or not really good at all of them…
- And finally there are those who might fit any of the characteristics above, but would have preferred not to listen their parents and rather open that scuba-diving shop in Honolulu.

Available Certifications

This diversity explains the existence of more common product-specific certifications: you can be certified to use Microsoft technologies (http://www.microsoft.com/learning/mcp/default.mspx), MySQL databases (http://www.mysql.com/certification/), Apple servers (http://www.apple.com/xserve/raid/certifications.html), various IBM products (http://www-03.ibm.com/certify/certs/index.shtml), Java development stacks (http://www.sun.com/training/certification/java/index.xml), Cisco routers (http://forums.cisco.com/eforum/servlet/CCNP?page=main), RedHat Linux installations (https://www.redhat.com/training/certification/) or UML diagrams (http://www.omg.org/uml-certification/index.htm)

However, given that technology companies have interest in having many people taking their certifications, their affordability and low-entry barriers to get them, many of these become much easy to get than they should be, and as a result, they lose credibility, and do not help IT recruiters to filter properly software developers during the selection processes. I've heard many complaints of project managers regarding these certifications, and I think it's a generalized feeling:

> "Certified skills pay has not just flat lined, it's in the negative. This is big news if you're certified and you're thinking about getting recertified," said Foote. "This trend is in the fourth quarter, that pay for certifications is on the wane, while non-certified skills are growing in pay."(...) Certifications are losing value because employers are looking for more in their workers than the ability to pass an exam; they want business-articulate IT pros."

(Rothberg, 2006)

Bruce Schneier, a well-known security researcher, has written about security certifications as well, with a mixed feeling:

> In the end, certifications are like profiling. They work , but they're sloppy. Just because someone has a particular certification doesn't mean that he has the security expertise you're looking for (in other words, there are false positives). And just because someone doesn't have a security certification doesn't mean that he doesn't have the required security expertise (false negatives). But we use them for the same reason we profile: We don't have the time, patience, or ability to test for what we're looking for explicitly.

(Schneier, 2006)

The conclusion of all of this is that the debate is pretty much still open, and that there is not a simple answer to it.

Market Fragmentation

There is an interesting anonymous comment in Schneier's website as well:

> Another thought on certification is they are not all equal.
>
> There are Vendor Certs. Microsoft's MCP/MCSE, CISCO CCNA/CCNP/CCIE Pro: The canidate is likley to know how to work on your specific platform. Con: The canidate is likely to think in only the vendor's interest.
>
> There are Certs to assure knowledge of standard security terminlogy. ISC CISSP Pro: Can talk strategy and evaluate the nine domains to evaluate how the company is doing overall Cons: Most likely could not tell you what the nineth byte of an ip packet means or if OpenSSL is out of date on Red Hat Linux.
>
> Topic specific, vendor neutral. SANS GIAC Pro: Vendor neutral. A lot of focus on specific skills in NIDS or Hardening Windows, Incident Handeling, etc. Con: Concentration on open source tools since they are easily available, but it does not seem to impress all employers.

(@nonymou5, in Schneier, 2006, spelling mistakes not corrected)

I think that this comment summarizes pretty well another problem with certifications: there is a great level of fragmentation in today's market. Every single important technology in the IT world requires a huge investment in time and practice in order to master it, and this translates in a huge complexity for the developers to choose the right certification. All of these without taking into account the large number of IT-related university degrees available, online or not.

Conclusion

The term "software engineer" is sufficiently vague, and the number of "certifications" sufficiently large, as to allow a single "yes or no" answer to whether professionals in the software sector should be certified or not. I personally think that I would rather avoid vendor-specific certifications as far as possible, and choose university-related or problem domain related certifications instead, to keep my career options open, and my mind free of marketing.

References

Rothberg, Deborah; "Another Nail in the IT Certification Coffin", Novembrer 3rd, 2006, [Internet] http://www.eweek.com/article2/0,1895,2051272,00.asp (Accessed June 3rd, 2007)

Schneier, Bruce; "Security Certifications", July 20th, 2006, [Internet] http://www.schneier.com/blog/archives/2006/07/security_certif.html

(Accessed June 3rd, 2007)

Shore, Julie; “Why Certification? The Applicability of IT Certifications to College and University Curricula”, [Internet] http://www.developer.ibm.com/university/scholars/certification/ebusiness/pdfs/why-certification.pdf (Accessed June 3rd, 2007)

Adding Manpower

Adrian Kosmaczewski

2008-08-08

Published in 1975, “The Mythical Man-Month” is considered an all-time classic in the software engineering field. The book author, Frederick P. Brooks Jr., used his experience as the project manager of the IBM System/360 and its software, the Operating System/360, to explain a common set of problem patterns, applicable to other software projects as well.

One of the most famous citations in the book is the one regarding the consequences of adding human resources to a late project; this article will provide a couple of thoughts about this assertion, and highlight some contrariwise opinions.

The Mythical Man-Month

The second chapter of Brooks’ masterpiece is named exactly as the book, “The Mythical Man-Month”; the core argument of this chapter is that the most frequent factor of project failure is schedule and time estimation. Brooks states that this is due to the fact that

> Men and months are interchangeable commodities only when a task can be partitioned among many workers **with no communication among them.** This is true of reaping wheat or picking cotton; it is not even approximately true of systems programming. When a task cannot be partitioned because of sequential constraints, the application of more effort has no effect on the schedule. The bearing of a child takes nine months, no matter how many women are assigned.

(Brooks, pages 16 & 17)

The final phrase of the above paragraph is often used as a graphical depiction of the nature and meaning of Brooks’ law. It implies the strong need for communication and integration existing in software projects; being social processes, software requires a strong network of communication between team members, allowing them to coordinate the inherent set of interdependencies that every project has.

After an interesting analysis of common time overrun situations, Brooks ends this chapter with the following conclusion, which

contains the enunciation of the law itself:

> Oversimplifying otrageously, we state Brooks's Law: **Adding manpower to a late software project makes it later.** This is then the demythologizing of the man-month. The number of months of a project depend upon its sequential constraints. The maximum number of men depends upon the number of independent subtasks. From these two quantities one can derive schedules using fewer men and more months. (The only risk is product obsolescence.) One cannot, however, get workable schedules using more men and fewer months. More software projects have gone awry for lack of calendar time than for all other causes combined.

(Brooks, pages 25 & 26)

This "law" is known and cited throughout the industry as an example of a common pattern, observed once and again in different projects all over the world:

> **Fact 3: Adding people to a later project makes it later**(…) Intuition tells us that, if a project is behind schedule, staffing should be increased to play schedule catch-up. Intuition, this fact tells us, is wrong. The problem is, as people are added to a project, time must be spent on bringing them up to speed.(…) Furthermore, the more people there are on a project, the more the complexity of its communication rises.

(Glass, page 16)

As a personal experience, I must say that the lecture of this book opened my eyes more than many, many other books. It is a funny read, but also an enlightening one: many anecdotes told by Brooks strangely correspond to my own experience, and this one is no exception. I have seen projects gone unfortunately late because of the simple fact of adding more people; and in one particular case, the project was cancelled altogether. These projects had several factors in common, though:

- **Bad documentation, or lack thereof;** the only way for newcomers to the project to know what was going on was interrupting the other developers, disrupting the current operations on the project; I think that a good set of documents, describing both the high-level architecture and the low-level APIs are needed for new developers to jump in and catch up. It's maybe not enough, but a good leap forward anyway.
- **Lack of architectural vision;** projects that do not have an architect, providing vision and technical leadership to the team, are in my opinion exposed to problems when more developers join the project. The architect can act as a proxy person, guiding new developers while they familiarize themselves with the

project, isolating other developers from this task.

- **Bad project decomposition in components;** if the system to be developed is sufficiently large, and the decomposition in components is not properly done, the overlap and extended communication paths among team members might affect the whole project negatively. A good decomposition breaks down the whole project in a set of smaller ones, with the corresponding set of interfaces, which brings the whole team to work separately on different subsystems. In these, the risk of getting later for adding manpower is reduced proportionally.
- **Bad working conditions;** I positively think that open spaces are a common disease in our industry. Teams working in open spaces suffer more of noise and visual distractions, and this is more evident when new team members join the project.

Criticism

However famous, Brooks' law has had a good deal of criticism as well, regarding the specific characteristics of projects that might be affected in case that new people is assigned to them. The OS/360 project, which served as the basis for Brooks' work, might not be similar to other projects, and as such, the law would not necessarily apply to them:

> For Brooks' Law to be true, the amount of training effort required from existing staff must be significant. The amount of effort lost to training must exceed the productivity contributed by new staff when they eventually become productive. (…) "Late" chaotic projects are likely to be much later than the project manager thinks–project completion isn't three weeks away, it's six months away. Go ahead and add staff. You'll have time for them to become productive. Your project will still be later than your plan, but that's not a result of Brooks' Law. It's a result of underestimating the project in the first place.(…) Controlled projects are less susceptible to Brooks' Law than chaotic projects. Their better tracking allows them to know when they can safely add staff and when they can't. Their better documentation and better designs make tasks more partitionable and training less labor intensive. They can add staff later in the project with less risk to the project.

(McConnell, 1999)

Scott Berkun gives a more concrete analysis on why the law could be wrong:

> - **It depends who the manpower is.** The law assumes that all added manpower is equal, which is not true.
> - **Some teams can absorb more change than others.** Some teams are more resiliant to change.

- **There are worse things than being later.** (…) That can be ok if you also get higher quality
- **There are different ways to add manpower.** (…) The more experience everyone has with mid-stream personnel changes, the better.
- **It depends on why the project was late to begin with.** (…) no amount of programming staff modifications will resolve the psychiatric needs of team leaders or the dysfunctions of executives.
- **Adding people can be combined with other management action.** (…) if you're removing your worst, and most disruptive, programmer and adding one of your best, it can be a reasonable choice.

(Berkun, 2006)

And what about open source projects? Many of these (Linux, Apache, MySQL) are potentially among the biggest software projects ever undertaken, and they don't appear to suffer o fthe problems pictured by Brooks' law:

> But proponents of open source and free software development, including Linux developers, are not completely satisfied with the Law. Most famously (among geeks at any rate), Eric Raymond in his "The Cathedral and the Bazaar," declared Brooks' Law obsolete, if not simply limited, saying "if Brooks' Law were the whole picture, Linux would be impossible." Although Raymond now says that he has somewhat modified his views or was misunderstood, some still would say he is given to oversimplifying and outrageousness himself. "I don't consider Brooks' Law 'obsolete' any more than Newtonian physics is obsolete; it's just incomplete. Just as you get non-Newtonian effects at high energies and velocities, you get non-Brooksian effects when transaction costs go low enough. Under sufficiently extreme conditions, these secondary effects dominate the system – you get nuclear explosions, or Linux."

(Jones, 2000)

Conclusion

So far, the discussion seems to be open. There might be a scale factor for projects, which in turn might expose them to be affected by Brooks' law. I think that research is needed to arrive to a conclusion, even if it will be a statistical one.

Other important facts highlighted in the book are the "second system phenomenon", the productivity advantage of using high-level languages, and the importance of building a prototype - "one to throw away". I can only recommend this book to everyone interested in the

field of software engineering (which I did in my own review of classic books in this blog:/blog/my-bookshelf-part-iii/)

References

Berkun, S.; “Exceptions to Brooks’ Law”, January 11th, 2006, [Internet] http://www.scottberkun.com/blog/2006/exceptions-to-brooks-law/ (Accessed June 8th, 2007)

Brooks Jr., F. P.; “The Mythical Man-Month - Essays on Software Engineering, Anniversary Edition”, 1995, Addison Wesley, ISBN 0-201-83595-9

Glass, R. L.; “Facts and Fallacies of Software Engineering”, Addison-Wesley, 2003, ISBN 0321117425

Jones, P.; “Brooks’ Law and open source: The more the merrier?”, IBM, May 1st, 2000, [Internet] http://www.ibm.com/developerworks/linux/library/os-merrier.html (Accessed June 8th, 2007)

McConnell, S.; “Brooks’ Law Repealed?”, IEEE Software, November/December 1999 [Internet] http://stevemcconnell.com/ieeesoftware/eic08.htm (Accessed June 8th, 2007)

Saving a Failing Project

Adrian Kosmaczewski

2008-08-11

In 2006 I had the opportunity to work as a "project leader" into a small failing project. Three developers were working in an ad hoc basis, creating a software application for an important client (a government office in Lausanne), without any kind of detailed formal specification, without any kind of design documentation, and with strong pressure from the management to release the application, even if not in an usable state. Needless to say, the project was also beyond budget.

I had just joined this company a couple of days ago, and the management asked me to take the project in charge. Not an easy task, particularly because it was my first experience of this kind.

The client was pushing to get the software it had paid for (it was a desktop reporting application for the Police department), and had not got any kind of preview yet. So the first thing I did is to pick up my copy of "Leading a Software Development Team" book and read chapter 2, "I'm taking over the leadership of an existing project // where do I start?" and get a thorough read:

> The first thing that you should start to do is to review the situation. This involves more than just absorbing impressions; you need to organize these impressions into a framework. Try to organize your thoughts into the following areas, and in each area try to separate technical issues from personnel ones:
>
> - Where is the team now? (…)
> - Where is it supposed to be getting to? (…)
> - How does the team currently intend to continue?

(Whitehead, page 17)

Another highly pragmatic resource was Joel Spolsky, and his "Joel Test":

> The neat thing about The Joel Test is that it's easy to get a quick yes or no to each question. You don't have to figure out lines-of-code-per-day or average-bugs-per-inflection-point. Give your team 1 point for each "yes" answer.(…)

> A score of 12 is perfect, 11 is tolerable, but 10 or lower and you've got serious problems. The truth is that most software organizations are running with a score of 2 or 3, and they need serious help, because companies like Microsoft run at 12 full-time.

(Spolsky, 2000)

The "Joel Test" result for this team was 2 when I joined the team (they just had source control and good tools). When I left the company, they were running at 9 (we just did not have candidates writing code during interviews, nor testers, nor hallway usability testing).

For this project I took the following decisions:

- Since the priority for the client was to see results, I asked the developers to concentrate on stabilizing "visible" features, particularly on a visual report editor, that used a complex set of controls, similar to those of a drawing application, to create reports. Doing this, we could have a stabilized preview version that we showed to the client as early as one week after my arrival to the project.
- In agreement with the developers, we set up a daily build procedure, and I also asked them to provide a "client build" every Wednesday, that would be placed in a public directory available to the client. It turns out that the client never downloaded the binaries, but they liked to see the version numbers grow, and the binaries being delivered. Every week, Wednesday was the "public build" day, Thursday was the "bug correction" day, and Friday, Monday and Tuesday were "new features day". Small stand-up meetings every day allowed us to know what was going on.
- Another important concern from the developers' side was to have a quiet environment to work. They were constantly interrupted by the (quite nervous) managers to see their progress, and as such, I decided to stand in between both; I asked them to not to interrupt the developers for any reason, and to ask me for updates. I became a "proxy" between both, which reduced the tensions, and brought some peace to the developers.
- I created a fast project plan in our Intranet (there wasn't any, so tracking the project was next to impossible) by asking the developers about the tasks they needed to do to finish the project, with the estimated time to do them, and setting some milestones. Since the project was in a wiki page, the developers could change the time estimations in case that they felt they had made a mistake; the only condition being to notify me of these changes.
- Using that information, I could create a couple of reports for everyone to see, and bring more visibility to the project:
  1. I wrote a weekly report stating the week's achievements, the status of the project (number of open bugs, new functionality available, etc).
  2. In the intranet, I set up a couple of graphs and report tables, which were automatically updated every day.

- I did not take any technical decisions about the project; I gave full authority on this matter to the lead developer, who in turn appreciated this trust and took spontaneously the decision of documenting and unit testing the system thoroughly doing extra hours every day. This boosted the morale of the team, and the quality of the application as well. The other two developers contributed to these tasks as well, and the rhythm of releases and their quality increased in a couple of months. It turns out that the architecture of the system was particularly well done, and as such, adding new features was a relatively simple task, once the underlying framework was done; of course, during that time no visible results were available, which made everyone nervous.

Looking backwards, the only technical decision I've needed to take during this project was to use the company wiki; there I could add information pages that everyone contributed to, reducing the number of communication channels and reducing the misunderstandings between project team members. I cannot stress how much this helped; it provided a complete dashboard for everyone to refer to.

The most important problems in this project were human and customer related ones. By providing more visibility to the project, and by reducing the signal-to-noise ratio in the communication channels between developers and management, the team was able to provide the customer with a more reliable and full-featured product.

Joel Spolsky, "The Joel Test: 12 Steps for Better Code", Wednesday, August 9th, 2000 [Internet] http://www.joelonsoftware.com/articles/fog0000000043.html (Accessed June 8th, 2007)

Whitehead, R.; "Leading a Software Development Team - A Developer's Guide to Successfully Leading People & Projects", Addison-Wesley, 2001, ISBN 0-201-67526-9

Dangers of Prototyping

Adrian Kosmaczewski

2008-08-15

Frederick P. Brooks Jr. has written about prototypes, saying that they are not only useful but strictly fundamental pieces of the overall software process, as in many other engineering activities. He gives the example of a pilot chemical plant, prepared to process 10'000 units per day instead of the 2 million units a day that the final plant would have to handle, in order to demonstrate the feasibility and uncover some unforeseen problems.

He summarizes his opinion in the famous phrase "plan to throw one away" (Brooks, 1995, page 116), underlining the problem of **change management**: managing change, right from the beginning of the project, instead of ignoring or avoiding it, is particularly important in software projects, since it presents a solid mindset for all stakeholders in order to avoid scope creep, schedule and staffing problems.

One of the proposed solutions to manage change (the "only constant thing") in a software project is the use of prototypes, which consist of UI mockups, showing basic interactions between the components and screens, allowing users to see in real time how the final software will look like, and serving as part of the feasibility study before creating the final product.

Several authors have highlighted different problems related to prototypes:

> We consider three dangers to be most serious.
>
> - Assumptions may be hidden (…)
> - Feedback may be too expensive (…)
> - Inappropriate human-computer interaction issues are highlighted

(Atwood et al, 1995, page 180)

> One of the risk of creating a User Interface Prototype is accidentally raising unrealistic expectations about future progress on the project.

(McConnell, 1998, page 117)

> Firstly, there are still plenty of users and managers around who think that the software is not much deeper than the

> user interface, and so if they see a nearly finished interface, they think the project is nearly done. (…)
>
> Secondly, engineers often get carried away about making a lovely user interface, and spend far too much time on it. (…)
>
> Lastly, and perhaps most importantly, is the fact that almost no code that is written as a 'temporary throw-away' actually gets thrown away. It usually gets used in the product in one way or another.

(Whitehead, 2001, page 254)

- Needs cooperation of management, developers, and users
- Managers may view prototyping as wasteful
- Managers and/or customers and/or marketing may view prototype as final product
- Programmers may lose discipline
- Prototype can be overworked (reason for prototype is forgotten)
- Prototyping tool may influence design
- Possibility of overpromising with prototype

(Hartson, 2001, page 5)

In my personal experience, I must say that I have not worked in many teams using prototyping extensively. Many times, when I have came up with the idea of doing a prototype, my managers (and even my coworkers) would dismiss it with the following criteria:

"We don't have time - money - staff - (add your own preferred variable here)"

"OK, but do the prototype in such a way that we could reuse it later"

"We do not lose time in prototypes in this company; we do real work here"

"We do not have a budget for prototypes in the project"

"We know exactly what the customer wants"

"Oh, we've tried that once, but the customers never like what they see anyway, so why bother"

"Our client does not want prototypes, period"

"Our CEO does not want prototypes, period"

"I do not want prototypes, period"

"Our policy does not allow prototypes"

However, I must add that in just one case (a successful project I worked in three years ago), the analysts team managed to fit in the

project budget the capability to create a set of user interface prototypes using **Microsoft Visio**. This tool (or any other diagramming tool) has many advantages over the traditional "Visual Basic"-based prototyping:

- Any user with Visio (be it analysts, developers, end users with basic Microsoft Office knowledge) can contribute to the prototypes, suggest changes and play with the mockups;
- No code is needed, which avoids developers to reuse it later;
- The output can be inserted (typically as images) in other documents, e-mails, etc.

This technique had a tremendously positive impact in the project:

- A great deal of feedback from the client was related to the mockups;
- The design documentation had an excellent complement of information, in the form of screenshots, coupled with the textual indication of every possible action on the user interface;
- The developers could use these indications to reduce the uncertainty and deliver a product that matched the final decisions exactly;
- Since the prototypes were not "reusable" (they were after all only drawings on Visio) the developers could not be tempted to reuse the code in the final product.

In my opinion, this approach was one of the key factors of success of the project, but not the only one: the team members and the customer agreed that the project was critical and complex, and that a prototype could help everyone to understand it better. There was a **specific mindset** in the whole project, at both sides of the equation, which made prototyping a good option, and a success factor. Without this mindset, I do not think that prototyping will fit in any project.

References

Atwood, M. E.; Burns, B.; Girgensohn, A.; Lee, A.; Turner, T.; Zimmermann, B.; "Prototyping Considered Dangerous", Interact '95 Conference Proceedings, Pp. 179-184, NYNEX Science and Technology [Internet] http://www.fxpal.com/people/andreasg/interact95-2.pdf (Accessed June 24th, 2007)

Hartson, "Rapid Prototyping and Its Role in Development and Evaluation of User Interaction", CS 5714 Fall 2001 - Usability Engineering, Virginia Tech [Internet] http://courses.cs.vt.edu/~cs5714/fall2001/notes/pdf/05_RP.pdf (Accessed June 24th, 2007)

Brooks Jr., F. P.; "The Mythical Man-Month - Essays on Software Engineering, Anniversary Edition", 1995, Addison Wesley, ISBN 0-201-83595-9

McConnell, Steve, "Software Project Survival Guide", Microsoft Press, 1998, ISBN 1-57231-621-7

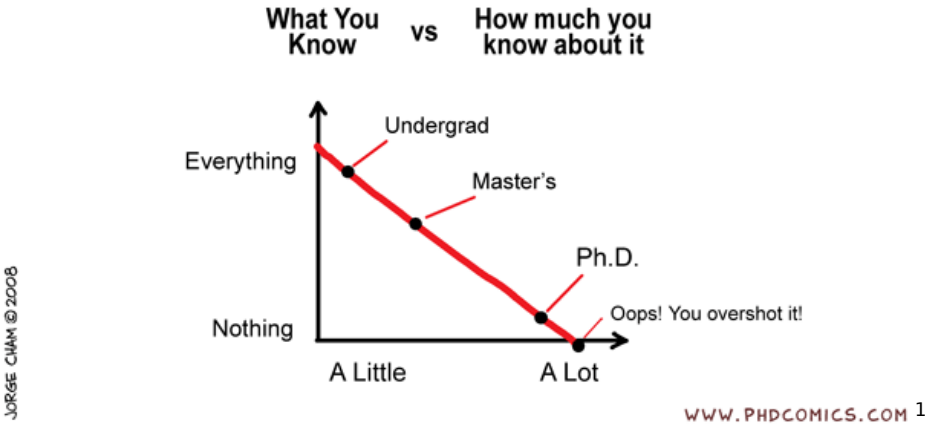
Whitehead, R.; “Leading a Software Development Team - A Developer’s Guide to Successfully Leading People & Projects”, Addison-Wesley, 2001, ISBN 0-201-67526-9

Master

Adrian Kosmaczewski

2008-08-28

I've sent the final version of my dissertation to the University of Liverpool. I've been doing this Master's degree since 2005, and now it's over. It feels good and weird at the same time.



[1]

Rem 1.0[2], the final result and main objective of my Master's thesis work, has been released today. A small, simple, yet extensible and portable UML tool written in C++, using Juce[3], POCO[4] and SQLite[5]. Not perfect but extensible and small. And that's what's great about it.

Update, 2023-05-26: Almost 15 years later, you can read my thesis here[6].

[1] http://www.phdcomics.com/comics/archive.php?comicid=1056
[2] http://remproject.org/
[3] http://www.rawmaterialsoftware.com/juce/
[4] http://pocoproject.org/
[5] http://www.sqlite.org/
[6] /books/MSc_Thesis/uliv_dissertation.pdf

iPhone Conference 2008 Geneva

Adrian Kosmaczewski

2008-09-15

Those of you who have been following this blog for the past years know that I have somewhat reduced my “writing rhythm” these days, and many factors have caused this. For the past 3 months I’ve been not only finishing my Master’s degree, but I started working as a full-time, independent iPhone developer, and soon some of my code will be available on the AppStore.

But in the meantime, I’m also starting a new business, and I’ve seen that many companies want to get into the iPhone application business in one way or another, and they have many questions.

To answer most of them all at once, and without breaking any NDA, we want to introduce you to The iPhone Conference 2008. This event is targeted to decision makers: whether you’re a CIO, CEO, marketing manager, responsible of corporate communications, we’ve got a message for you.

This event will be the first in Switzerland targeting the iPhone (that we’re aware of, of course). It will be held in Geneva, at the CICG, and it will be held in English. I think that there are huge business opportunities opening right now with the iPhone; I hope to meet you there!

Interview sur com.in Magazine

Adrian Kosmaczewski

2008-09-30

Interview by Victoria Marchand, appeared in the October 2008 issue of com.in magazine[1] (download PDF scan of the article[2]).

> – Adrian Kosmaczewski, vous vous définissez comme un “technologiste”, comment en êtes-vous venu à vous intéresser à l’iPhone? – Parce que cet objet, qui est bien plus qu’un téléphone, marque une nouvelle étape dans l’histoire technologique, à l’instar des premiers Mac, des écrans couleurs et des télephones mobiles. – L’iPhone est pourtant avant tout un mobile? – C’est beaucoup plus que cela. Certes, on peut téléphoner, mais cet appareil a d’autres fonctionnalités qui permettent d’informer, de surprendre, de divertir et de fournir, notamment par le biais d’applications, toutes sortes de services.

[1] http://www.cominmag.ch/

[2] /press/com_in_iPhone_conf.PDF

iPhone Conference 2008 Teaser 1

Adrian Kosmaczewski

2008-10-08

Teaser #1[1] for the iPhone Conference[2] in Geneva.

[1] https://www.youtube.com/watch?v=J_nBu6ZBn6s
[2] https://web.archive.org/web/20081011200352/https://www.iphone-conference.ch/

iPhone Conference 2008 Teaser 2

Adrian Kosmaczewski

2008-10-09

Teaser #2[1] for the iPhone Conference[2] in Geneva.

[1] https://www.youtube.com/watch?v=NcxbSdmzQX8
[2] https://web.archive.org/web/20081011200352/https://www.iphone-conference.ch/

iPhone Conference 2008: a bit of magic!

Adrian Kosmaczewski

2008-11-03

This is the talk[1] I gave last Friday during the first edition of the iPhone Conference 2008[2]! I hope you'll enjoy it as much as I enjoyed giving it :)

how to put your company in people's pocket? from Maël Guillemot on Vimeo.

[1] https://vimeo.com/2129015
[2] https://web.archive.org/web/20081011200352/https://www.iphone-conference.ch/

iPhone Font Browser

Adrian Kosmaczewski

2008-11-12

One of the most common questions I get when working with clients on iPhone apps is this one:

> Which fonts can I use in iPhone apps?

I was surprised to see that the Xcode documentation does not bring a list of the fonts available in the iPhone OS (have I missed it? Is it somewhere else?), and I was happy to find this blog post the other day. Taking his idea as a basis, I've created a font browser for the iPhone which I've published in the projects section of this blog.

Feel free to play with the code! As usual, no warranties of any kind, but I'm already using it every day :)

The Dirty Little Secret of iPhone Development

Adrian Kosmaczewski

2008-12-23

This is happening right now, at a web agency near you.

The dot-com boom of the 90's spawned a brand new generation of coders and software developers, including me, by the way. While before that time the term of "software developer" might have been reserved to system programmers fluent in C, COBOL, C++ or other languages, right now the vast majority of developers I know spend their time writing web applications, either public or in a private intranet, in J2EE, ASP.NET, Rails, PHP, you name it.

I have said before[1] that writing web applications should be taken as seriously as writing desktop systems. Call me names if you want, but I'm a big fan of Joel's Test[2].

However, after all this years, after the Chaos reports[3], after Peopleware[4], after the Mythical Man Month[5], people still treat quality[6] as an afterthought. And also complain about how much software sucks, how expensive it is, and how late it arrives, by the way. Now that the iPhone SDK is widely available, that the App Store is selling more apps that we could have had imagined 6 months ago, many web agencies want to jump to native iPhone development contracts, which are hype and nice and pricey and whatnot. Which is only going to make things worse.

The dirty little secret in this story is this: **iPhone development looks more like developing applications for a desktop operating system, and less, much less than web development.** And I'm frightened to see some small shops (and even bigger ones), who never attained a real level of professionalism or quality in their software tasks, starting projects and realizing, later, when they are over budget and behind schedule[7], that this kind of applications requires a different

[1] /blog/web-development-equal-software-development/
[2] http://www.joelonsoftware.com/articles/fog0000000043.html
[3] http://www.cs.nmt.edu/~cs328/reading/Standish.pdf
[4] /blog/my-bookshelf-part-iii/
[5] /blog/adding-manpower/
[6] /blog/software-project-quality/
[7] /blog/schedule-issues/

mindset.

Let's repeat something that you should know by now: web apps are easy to maintain. To begin with, you just have one instance running of it, and as Paul Graham said[8], you can write them in any language you want, you can release bug fixes with your application running, monitor performance issues live, change its appearance quite easily and for everyone at once, etc. This is probably the single biggest advantage of web apps. With AJAX we even got the possibility of going beyond in terms of user interface, responsiveness, perceived speed, you name it, and today the number of famous web apps is outstanding.

However, web apps run into this terrific (and terrifying) sandbox called the browser. They (normally) cannot access your file system, they cannot open other applications and interact with them – well, with some custom URL schemes like lastfm: mailto: or skype: you might have the ability to do some things, but certainly not much more than activating the application in some limited way as a help for the user. You cannot access the user's hardware directly – well, again, you can access the webcam through Flash, or you can trigger the window.print() method to have the native print dialog pop up, but that's more or less the maximum you can do.

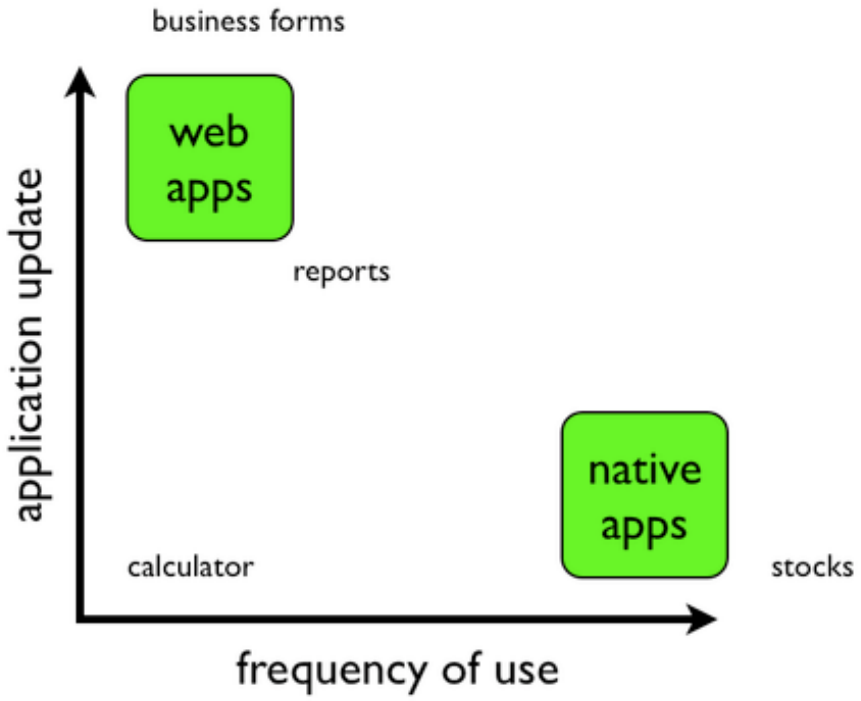
In the iPhone, there is a similar situation. You choose to create a native iPhone application over a web one when you require one or many of these things in your application:

- GPS geographical data;
- Accelerometer information;
- Photo camera or library;
- 3D graphics;
- Complex animations;
- Address Book entries;
- Sound recording and playback;
- etc…!

I've talked about the dichotomy between iPhone native vs. web apps in my speech during the iPhone conference[9] where I used this graphic, which might help those having to decide whether they should do a native or a web application:

[8] http://www.paulgraham.com/avg.html

[9] /blog/iphone-conference-2008-a-bit-of-magic/



The tradeoff, basically, consists in knowing that having the ability to access these features, means that you are giving up your capacity to roll out quick upgrades to your code. You have to depend on Apple's own review process timings for that, which, as far as I've seen so far, cannot be predicted. And finally, you depend on the user's final will to update your application!

Similarly, you also lose the ability to create these applications using your common, well-known, garbage-collected programming language, but you'll have to deal with Objective-C exclusively, together with its weird syntax, possible memory leaks, eventual out-of-memory notifications, lazy loading techniques, compiler warnings and errors, and seeing the MVC design pattern even in your wildest dreams at night. Regarding the syntax, I admit that I love it, but it took me a while to get used to it, when I started playing with it back in 2003.

Given these conditions, when you start an iPhone project you will have (let me be very clear about this, so I'll repeat) you will have to dust out your good old desktop application programming techniques (or learn if you don't have them), read Code Complete again (which I'm doing right now) and take all the advice that you can from C and C++ programmers who have been working in this kind of stuff for the past 20 years.

I've even done my Master's degree project in C++[10] because of this:

[10] http://remproject.org/

the mindset required for iPhone apps is not the same as for your day-to-day web application, and is closer to that of a desktop application. I've been doing web apps for 12 years now, and desktop apps for 5 years (mostly in C#, Objective-C and C++); that's my ground for stating this. You require a spec, you require tests, you require testers, you require daily builds, you require bug databases, you require quiet working conditions[11].

You require at least an 11 in Joel's Test[12] to know that you're doing fine, but not only that: you need to know about pointers, you need to know C[13], you need to know that your code runs in an fragile environment with EXC_BAD ACCESS (SIGBUS) and low-memory warnings and stuff like that happening when you expect it the least.

Unfortunately, from what I've seen so far (at least here in the Lake Geneva region) many companies are spending much more than they intended to for rolling out their iPhone apps; these are real quotes from people I've dealt with in the past 6 months:

- "Oh, we do not write specs, we prefer to modify the code as we have ideas!";
- "Oh, we do not write tests, we do not have the time for that!";
- (in general) "Oh, we do not work like that here. We do what the client asks and that's all.";
- "We will not follow Apple's iPhone Human Design Guidelines at all; we have our own. Users must double-click on buttons, so they feel more comfortable while using our application";
- "We want our [PUT YOUR FAVORITE UIKIT CLASS NAME HERE] to have [SOME IMPOSSIBLE FEATURE WHICH DEFIES GRAVITY]. Easy, right?"
- "We have to remove this XYZ feature, right now!", which means rewriting half of the code and leads to this: "What? So much? For such a small change?"
- "Why have you spent all that time 'fixing memory leaks'? I won't pay for that. Please concentrate in the new features we've asked. By the way, what are memory leaks?"
- "Here's the styles document you asked, with the colors and fonts for the application" and the guy provides me with… a CSS stylesheet. Which references a font not available on the iPhone[14], by the way, and with colors in hex format.
- "We want an animation at startup, like the Chanel application" (everyone wants to release the next Chanel iPhone app these days) "here's an [animated GIF / Flash movie / PowerPoint slides] with the animation for you."
- "Our team has slightly modified the code when you weren't there, but it does not work any more, could you fix it please?"

[11] /blog/open-space-or-individual-offices/
[12] http://www.joelonsoftware.com/articles/fog0000000043.html
[13] http://cocoadevcentral.com/articles/000081.php
[14] /blog/iphone-font-browser/

And of course, the all-time winners: the support tickets stating that "everything is slow" without further indication, or that "the application hangs", without any details in it.

OK, I confess, I'm a bit of an extremist here; many of the quotes above are valid in some contexts. But not those of the small-to-medium sized iPhone projects I've been involved in lately, with the urgency of releasing the code as fast as possible, just for the sake of being there, in the App Store, right now.

There's a long road ahead. The problem, again, is not the technology itself, but the people involved in these projects (me, for example :). There's a bit of what Joel mentioned a while ago, about the perils of Java schools[15], which actually only has to do with developers, but there's also the issue about teaching the clients the limits of the platform, and about creating a strong software engineering body of knowledge in companies which usually did not need it previously.

We've been doing web apps for so long that we've forgotten how to sit down, take a deep breath, fix that goddamn memory leak, and realize that, as Brooks and Joel said[16], good software takes time.

[15] http://www.joelonsoftware.com/articles/ThePerilsofJavaSchools.html
[16] http://www.joelonsoftware.com/articles/fog0000000017.html

No es tan asi

Adrian Kosmaczewski

2008-12-25

Dice Clarin:

> Según la ley europea, los bancos son los responsables de las inversiones de sus clientes y contra ellos los inversores damnificados atacarán judicialmente. Se espera una cadena de juicios multimillonarios cuando ya se habla del absoluto fracaso de los organismos de control en las grandes capitales financieras en medio de la crisis.

En Suiza, esto no es asi: ha habido varios damnificados, no solamente por el asunto Madoff, sino tambien por la quiebra de Lehmann Brothers, y **la ley Suiza no responsabiliza a los bancos por sus clientes sino hasta un monto de 30'000 francos suizos.** Estos bancos (el Credit Suisse y la UBS, pero tambien algunos bancos menores) proponen a sus clientes, de manera intempestiva y brutal, compensaciones por montos que varian entre el 20 y el 50% del monto originalmente invertido por sus clientes, y si aquellos no aceptan, el banco se reserva el derecho de no devolver nada de nada.

Ser banquero en Suiza es el mejor negocio posible: no solamente no tenes ninguna responsabilidad por tus clientes, sino que ademas, cuando te va mal, el gobierno te da 70 mil millones de francos suizos para que no te vayas al joraca. **Feliz Navidad!**

Best Books of 2008

Adrian Kosmaczewski

2009-01-06

You might remember my beloved mantras: **learning a new programming language**[1] **and reading at least 6 relevant books**[2] **every year**. Following the 2007 edition[3], here's the list of the 8 books I have enjoyed most in 2008, ordered by a purely subjective and absolutely irrational decreasing preference. I strongly recommend all of them!

Winner: Geekonomics: The Real Cost of Insecure Software[4] **by David Rice**[5]

Runner-up: The No Asshole Rule: Building a Civilized Workplace and Surviving One That Isn't[6] **by Robert I. Sutton, PhD**[7]

And 6 more:

- Software Estimation: Demystifying the Black Art[8] by Steve McConnell[9]
- Modern C++ Design: Generic Programming and Design Patterns Applied[10] by Andrei Alexandrescu[11]
- It's Not How Good You Are, It's How Good You Want to Be[12] by Paul Arden[13]
- RESTful Web Services[14] by Leonard Richardson[15] and Sam

[1] /book/a-new-programming-language-every-year/
[2] /blog/my-bookshelf-part-i/
[3] /blog/best-books-of-2007/
[4] http://www.geekonomicsbook.com/
[5] http://blog.geekonomicsbook.com/
[6] http://www.amazon.com/Asshole-Rule-Civilized-Workplace-Surviving/dp/0446526568
[7] http://bobsutton.typepad.com/
[8] http://www.stevemcconnell.com/est.htm
[9] http://www.stevemcconnell.com/
[10] http://www.amazon.com/Modern-Design-Programming-Patterns-Depth/dp/0201704315
[11] http://erdani.org/
[12] http://www.phaidon.com/Default.aspx/Web/its-not-how-good-you-are-its-how-good-you-want-to-be-9780714843377
[13] http://www.paularden.com/
[14] http://oreilly.com/catalog/9780596529260/
[15] http://www.crummy.com/

Ruby[16]
- JavaScript: The Good Parts[17] by Douglas Crockford[18]
- Programming Collective Intelligence: Building Smart Web 2.0 Applications[19] by Toby Segaran[20]

Geekonomics: The Real Cost of Insecure Software

My big winner for 2008. If you don't care about software quality (yet), or if you think that machines aren't already in charge or our world, this book is for you. This has been one of the most hyped releases of 2008, and indeed, it can send a chill down your spine. Geekonomics is a lively catalog of software glitches and disasters, showing users as "crash test dummies", describing a whole industry built upon "end user license agreements", removing liabilities as no other industry has ever been capable of. Even the world of free and open source software is described with strong criticism.

Geekonomics sometimes feels like a desperate plea to apply Deming[21]'s Total Quality Management[22] principles in the world of software.

The book is interesting in several fronts. David Rice pledges for the creation of standards of quality, as well as tightening the requirement for certification of practitioners. His views are based on the United States, describing the legal framework of "tort law", the economic foundations of "incentives" for industries, and using comparisons with other economic sectors, particularly with the automotive industry. The book is academic yet entertaining, frightening but instructive, but sometimes falling on the side of sensationalism, in my opinion. I could even say that some of Rice's proposals are downright impossible to achieve, given the particular characteristics of the software industry, and the situation of the legal system in other parts of the world.

In any case, even if I do not particularly agree with all of his views, David Rice is right to emphasize his point strongly, giving concrete proposals, and opening the debate: the book would not have had the same impact if it had been written mildly.

Geekonomics was enlightening: it made me think about the quality of my own work in a completely different way (and prompted me to talk about it at Lausanne's 2008 Barcamp). I think that taking a time of introspection, and reading your own code with different eyes is required to realize that we are, as software developers, responsible for much

[16] http://intertwingly.net/blog/
[17] http://oreilly.com/catalog/9780596517748/
[18] http://javascript.crockford.com/
[19] http://oreilly.com/catalog/9780596529321/
[20] http://blog.kiwitobes.com/
[21] http://deming.org/
[22] http://www.mftrou.com/edwards-deming.html

of what is going on in the world. And this is the major contribution of this book.

The No Asshole Rule: Building a Civilized Workplace and Surviving One That Isn't

Some of my former colleagues will chuckle when they see this entry, because I've flown from a previous job a couple of years ago because of a total, complete, utter, outright and unmitigated asshole. A complete control freak, self-proclaimed genius, irresponsible jackass, who was the reason why 5 people (including me) left the place in less than 3 months.

(Guys, feel free to leave comments below ;)

More seriously, in the software industry it's common to see that great developers are, more often than not, shy or introverted. This fact, coupled with the Swiss' legendary attitude of eternal conciliation and conflict avoidance, has the side effect of creating troubled workplaces all over the country. No wonder this book made the headlines[23] last month in Switzerland. This is a huge problem which is only being revealed right now.

In any case, this book is, together with Peopleware[24], a gem of team-building and a real bag of tricks to avoid turnover and burnouts in your company. An absolute read to everyone involved in the creation of workplaces, in every possible industry.

Software Estimation: Demystifying the Black Art

This book should be a mandatory read for every software developer, in every company, all over the world. It is a true gem, and a book that will become a timeless classic[25].

How many times have you been asked to estimate a project? To provide a deadline? To approve an estimation done by someone else? This book starts by enumerating the problems related to estimation, including those related to the definition of the word "estimation", the politics and the economics surrounding and defining what is accepted and required, and the common traps where all of us have stumbled upon at least once. Then it provides a catalog of common estimation methods, from the most obvious to the most complex ones, including a description of their relative drawbacks and benefits.

McConnell is the author of Code Complete[26] (which I'm re-reading these days) and Rapid Development[27] (which I plan to read soon).

[23] http://www.letemps.ch/emploi/affichearticle.asp?artid=246483
[24] /blog/my-bookshelf-part-iii/
[25] /blog/my-bookshelf-part-iii/
[26] a%20href=%22%22
[27] http://www.stevemcconnell.com/rd.htm

This guy knows what he's talking about, and he provides all the data required to support his claims. I cannot stress this more: if you are a project manager, a developer, a tester, an architect, or deal with software projects in some uncanny way, you **have** to read this book.

Modern C++ Design: Generic Programming and Design Patterns Applied

I bought this book during the writing of my Master's degree[28] dissertation project[29]. I had heard about it before, but since my project involved a lot of C++ template metaprogramming, I thought that the best idea was to check with the real experts. And boy, I'm happy I did.

This book goes beyond your common programming grounds, whichever your favorite language is. I personally enjoy writing C++ a lot, but that's me, and your mileage may vary. Doing multiple inheritance mixed with template metaprogramming[30], however, stretches your mind[31] into the realm of what you previously considered esoteric and impossible, and that's precisely the kind of readings that take you a long way forward.

If you are looking for a new way to write your old C++ code, or if you are asking yourself how different are C++ templates from Java or C# generics, read this book. You'll find a lot of interesting stuff in it, with explanations related to an existing library (written by the author) called Loki[32], which provides the implementation of several patterns explained in the book.

It's Not How Good You Are, It's How Good You Want to Be

This little book (less than 130 pages long) was written by Paul Arden, former creative director of Saatchi & Saatchi[33], a recognized advertising agency (whose "early growth was also helped by a policy of settling the invoices from small suppliers as late as possible, while promptly paying large, high-profile companies", dixit their Wikipedia article. No comments.)

Paul Arden provides extremely interesting and pragmatic views on creativity, people management, client relationship management and other stuff; if you happen to work in a web or advertising agency, you should read this book. However, as a software developer, I think that many of his views are still applicable to our own activity, particularly

[28] /blog/master/
[29] http://remproject.org/
[30] http://remproject.org/2008/05/26/activerecord-and-unit-tests/
[31] /blog/templates/
[32] http://loki-lib.sourceforge.net/
[33] http://en.wikipedia.org/wiki/Saatchi_and_Saatchi

when, like me, you're beginning your own independent path. It is not always obvious how to deal with situations when you're "in charge", and Arden's recommendations do help.

RESTful Web Services

For years I thought that SOAP was the way to go. And then some Rails activists[34] started talking about RESTful this, RESTful that, and well, it tickled my curiosity. It all started with Roy Fielding[35]'s doctoral dissertation about network architectures[36], followed by all the buzz of Web 2.0 APIs, which in the case of most startups took the form of RESTful APIs, found to be much easier to document, use and maintain than their XML-RPC or SOAP counterparts.

I read this book because of a requirement of a project where I worked at the beginning of the year, and this led to several[37] blog[38] posts[39] and even a project, that are still today quite popular. I've been using the RESTful approach for other projects, involving .NET, the iPhone and Cocoa on the Mac, and I would have never thought that doing network-based programming could be this fun. Looking backwards, attempts like SOAP and XML-RPC look clunky, adding layers of unneeded complexity for most projects, and creating a higher barrier of entry for new users of an API.

JavaScript: The Good Parts

Douglas Crockford is head of web development at Yahoo!, and probably the person in the world that knows most about JavaScript. I've been reading his articles[40] since I started working in my Propano project[41], and then watching his videos[42] as soon as they became available.

His views of JavaScript as the World's Most Misunderstood Programming Language[43] helped me see this (now I think) beautiful language under a new light. And of course, I could not miss reading his short (170 pages) but extremely detailed book. I thoroughly enjoyed it and strongly recommend its reading to anyone dealing with JavaScript in any way.

[34] http://rubyonrails.org/activists
[35] http://www.ics.uci.edu/~fielding/
[36] http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm
[37] /blog/django-rest-test-support/
[38] /blog/django-architecture-approaches/
[39] /blog/playing-with-http-libraries/
[40] http://www.crockford.com/
[41] ../sistema-propano/propano.zip
[42] http://video.yahoo.com/watch/630959/2974197
[43] http://javascript.crockford.com/javascript.html

Programming Collective Intelligence

I love programming books focusing on solutions rather than on specific features of some language; they provide insight on how to solve problems, showing the mathematical background required to do it. This is one of them.

This book can be considered a practical handbook on statistics programming, using Python for the code examples, but providing all the required insight and theory required to understand the logic beneath every code bit. The focus on Web 2.0 applications is clear, and this book has helped more developers than the author might ever know. Social sites are the realm of networks, large numbers, big datasets and complex relationships, and the techniques described in this book can help developers get out more information about their databases. Ever wondered how LinkedIn finds out someone you might know? This book has the answer for you.

What about you?

I would be very pleased if you could leave, in your comments below, the reference to your own preferred books. I am sure I have missed some gems last year, but 2009 is just starting!

Objective-C REST Client Update

Adrian Kosmaczewski

2009-01-19

I've uploaded (yet another) update to the Objective-C REST client I've blogged about previously. This time I've scanned the code with the excellent LLVM/Clang Static Analyzer and fixed a couple of memory leaks here and there. I strongly recommend to scan your own projects with this tool, it's extremely simple to use:

- Install it somewhere in your PATH;
- Set your projects to use the Debug configuration when building from the command line (you can do that in the inspector for the project, in the "Configurations" tab);

  (see Sebastien's comment below ;)
- Open Terminal.app and fire `scan-build -k -V xcodebuild` on the root of the Xcode project folder;
- If there are any problems with your code, you'll have your web browser pop up with the list of problems, their description in annotated code format, and even a link to open the file right away.

I have also fixed another problem with the code, which was preventing POST data to be properly sent to the server with the previous version. You might have encountered this problem, and here is the solution: instead of using NSString's stringByAddingPercentEscapesUsingEncoding: method, use CoreFoundation's CFURLCreateStringByAddingPercentEscapes() function. This means that this code:

```
[params appendFormat:@"%@=%@&", [key stringByAddingPercentEscapesUsingEncoding:
[[parameters objectForKey:key] stringByAddingPercentEscapesUsingEncoding:NSUTF8
```

became this:

```
NSString *encodedKey = [key stringByAddingPercentEscapesUsingEncoding:NSUTF8Str
CFStringRef value = (CFStringRef)[[parameters objectForKey:key] copy];
// Escape even the "reserved" characters for URLs
// as defined in http://www.ietf.org/rfc/rfc2396.txt
CFStringRef encodedValue = CFURLCreateStringByAddingPercentEscapes(kCFAllocator
            value,
            NULL,
            (CFStringRef)@";/?:@&=+$,",
            kCFStringEncodingUTF8);
```

```
[params appendFormat:@"%@=%@&", encodedKey, encodedValue];
CFRelease(value);
CFRelease(encodedValue);
```

Now any text sequence, including any “legal” URL character (as explained in the RFC), will be encoded properly and sent to the server as required.

Cocoa only provides a thin layer over many of CoreFoundation functions and types; it does not expose completely all the functionality “below”, and that’s why sometimes you must dig a bit deeper and call CoreFoundation code to certain operations, like using the Address Book data, or playing sounds in your iPhone applications. The good thing, as always, is that CoreFoundation types are “toll free” bridged, which means that you can safely cast a CFStringRef into an NSString pointer without any overhead.

Chris Adamson explained this as an “Opt-in Complexity” pattern, in an article in the Inside iPhone blog last week:

> Need time zone awareness? NSTimeZone is your friend. Need to know every time zone that the device supports? Get to know `CFTimeZoneCopyKnownNames()`. Again, a niche-ier feature lives down at the Core Foundation level, and isn’t wrapped by an equivalent call in Foundation, though it’s easy enough to switch to procedural C and make the one-off lower-level call.

10 iPhone Memory Management Tips

Adrian Kosmaczewski

2009-01-28

Memory management in the iPhone is a hot topic[1]. And since I'm talking about it on tonight's monthly meetup of the French-speaking Swiss iPhone Developers group[2], I might as well share some tips here from my own experience.

I won't go dive through the basics; I think that Scott Stevenson did a great job in his "Learn Objective-C" tutorial at CocoaDevCentral[3], from where the image below comes. I'm just going to highlight some iPhone-specific issues here and there, and provide some hints on how to solve them.

[4]

To begin with, some important background information:

- The iPhone 3G has 128 MB of RAM[5], but at least half of it might be used by the OS; this might leave as little as 40 MB to your application… but remember: you will get memory warnings even if you only use 3 MB;
- The iPhone does not use garbage collection, even if it uses Objective-C 2.0 (which can use garbage collection on Leopard, nevertheless);
- The basic memory management rule is: for every [alloc | retain | copy] you have to have a [release] somewhere;
- The Objective-C runtime does not allow objects to be instantiated on the stack, but only on the heap; this means that you don't have "automatic objects", nor things like auto_ptr objects to help you manage memory;

[1] http://www.mobileorchard.com/iphone-memory-management/
[2] http://www.facebook.com/group.php?gid=39755092833
[3] http://cocoadevcentral.com/d/learn_objectivec/
[4] http://cocoadevcentral.com/d/learn_objectivec/
[5] http://daringfireball.net/2008/10/iphone_3g

- You can use autorelease objects; but watch out! Since they are not released until their pool is released, they can become de facto memory leaks for you…;
- The iPhone does not have a swap file, so forget about virtual memory. When there is no more memory, **there is** no more memory.

Having said this, here's my list of tips:

- Respond to Memory Warnings
- Avoid Using Autoreleased Objects
- Use Lazy Loading and Reuse
- Avoid UIImage's imageNamed:
- Build Custom Table Cells and Reuse Them Properly
- Override Setters Properly
- Beware of Delegation
- Use Instruments
- Use a Static Analysis Tool
- Use NSZombieEnabled

Respond to Memory Warnings

Whatever you do in your code, please do not forget to respond to memory warnings! I can't stress this much. I have seen application crashes just because the handler methods were not present on the controllers, which means that, even if you do not have anything to clear in your controller, at least do this:

```
- (void)didReceiveMemoryWarning
{
    [super didReceiveMemoryWarning];
}
```

And you might as well respond to them on your application delegate, as follows:

```
- (void)applicationDidReceiveMemoryWarning:(UIApplication *)application
{
    [[ImageCache sharedImageCache] removeAllImagesInMemory];
}
```

For the description of the ImageCache class, continue reading ;)

Or finally, as an NSNotification:

```
NSNotificationCenter *center = [NSNotificationCenter defaultCenter];
[center addObserver:self
           selector:@selector(whatever:)
               name:UIApplicationDidReceiveMemoryWarningNotification
             object:nil];
```

Avoid Using Autoreleased Objects

Autoreleasing objects is easy and useful, but on the iPhone you should be careful with it. By default there is an NSAutoreleasePool instance created for you at the beginning of the main() function, ~~but this pool is not cleared up until your application quits! This means that during runtime, your autoreleased objects are de facto memory leaks, since they are retained until the application quits.~~ **(please see the comments below; I have experienced better performance when avoiding autoreleased objects, but my understanding of pools is misleading :)**

I started getting a better performance from my iPhone apps when I stopped using some methods creating autoreleased objects, for example:

```
// Instead of
NSString *string = [NSString stringWithFormat:@"value = %d", intVariable];

// use
NSString *string = [[NSString alloc] initWithFormat:@"value = %d", intVariable
...
[string release];
```

In version 2.0 of the iPhone OS there was also the problem that some "convenience methods" did not work at all; I'm sure you've experienced your application crashing when using NSDictionary's dictionaryWithObjects:forKeys: and then finding out that a replacing that with initWithObjects:forKeys: made your application run just fine. The NDA did not help at the time!

This does not mean that you can't use autoreleased objects; you should use them, for example, when you have factory methods returning objects not owned neither by the factory nor by the client calling it. You can also use autorelease pools in loops, when you need to allocate lots of small objects, but remember to release the pool right afterwards:

```
NSAutoreleasePool *pool = [[NSAutoreleasePool alloc] init];
for (id item in array)
{
    id anotherItem = [item createSomeAutoreleasedObject];
    [anotherItem doSomethingWithIt];
}
[pool release];
```

Remember to always release the pool in the same context where it was created. CocoaDev has an interesting discussion about using NSAutoreleasePools in loops[6].

Oh, and please, never release an autoreleased object on the iPhone: your application will crash almost instantly.

[6] http://www.cocoadev.com/index.pl?NSAutoreleasePool

Use Lazy Loading and Reuse

If your application consists of several different controllers embedded into each other, defer their instantiation until the last possible moment; this means in practical terms that your init method is minimalistic, and that you do more stuff when you need it; the example below is a typical list + detail layout, using a UITableViewController subclass inside a UINavigationController:

```
@interface UITableViewControllerSubclass
{
@private
    NSMutableArray *items;
    DetailController *detailController;
    UINavigationController *navigationController;
}
@end

@implementation UITableViewControllerSubclass

#pragma mark -
#pragma mark Constructors and destructors

- (id)init
{
    if (self = [self initWithStyle:UITableViewStylePlain])
    {
        // only basic stuff
        items = [[NSMutableArray] alloc] initWithCapacity:20];
        navigationController = [[UINavigationController alloc]
                                initWithRootViewController:self];
    }
    return self;
}

- (void)dealloc
{
    [items release];
    [detailController release];
    [navigationController release];
    [super dealloc];
}

// ...

#pragma mark -
#pragma mark UITableViewDelegate methods

- (void)tableView:(UITableView *)tableView
        didSelectRowAtIndexPath:(NSIndexPath *)indexPath
{
```

```
    Item *item = [items objectAtIndex:indexPath.row];
    if (detailController == nil)
    {
        detailController = [[DetailController alloc] init];
    }
    detailController.item = item;
    [self.navigationController
        pushViewController:detailController
                  animated:YES];
}

// ...

@end
```

As you can see in the example above, not only we are creating a DetailController instance only when needed (that is, when the user taps on an item in the UITableView), but we're reusing it every time the user taps on another cell of the table; this has another benefit: a reduction of object allocation and instantiation, which also helps increasing performance a little bit.

You could also use UIViewController's viewWillAppear: and viewDidDisappear: methods to perform some kind of lazy loading initialization and release, and if you really need to go further, you could use the UITabBarControllerDelegate's tabBarController:didSelectViewController: method to load and unload parts of your application from memory, as you need it.

Avoid UIImage's imageNamed:

Alex Curylo has written an absolutely great article[7] about the problems with UIImage's imageNamed: static method. It seems (and in my tests this appears to be true) that the iPhone OS (versions 2.0 and 2.1 at least) uses an internal cache for images loaded from disk using imageNamed:, and that in cases of low memory this cache is not cleared up completely (this seems to be corrected with version 2.2, though, but I cannot confirm).

Since I have projects that must run on version 2.0 of the iPhone OS, I have created a UIImage category with the following method:

```
@implementation UIImage (AKLoadingExtension)

+ (UIImage *)newImageFromResource:(NSString *)filename
{
    NSString *imageFile = [[NSString alloc] initWithFormat:@"%@/%@",
                           [[NSBundle mainBundle] resourcePath], filename];
    UIImage *image = nil;
    image = [[UIImage alloc] initWithContentsOfFile:imageFile];
```

[7] http://www.alexcurylo.com/blog/2009/01/13/imagenamed-is-evil/

```
    [imageFile release];
    return image;
}

@end
```

The name of the method includes the word “new”, to comply with Objective-C’s naming guidelines, since the object we’re returning to the caller is not autoreleased and has a retain count of 1. The caller is then owner of the UIImage and responsible to release it.

Once I have this UIImage instance, I place it in an image cache with this interface:

```
#import <Foundation/Foundation.h>

@interface ImageCache : NSObject
{
@private
    NSMutableArray *keyArray;
    NSMutableDictionary *memoryCache;
    NSFileManager *fileManager;
}

+ (ImageCache *)sharedImageCache;

- (UIImage *)imageForKey:(NSString *)key;
- (BOOL)hasImageWithKey:(NSString *)key;
- (void)storeImage:(UIImage *)image withKey:(NSString *)key;
- (BOOL)imageExistsInMemory:(NSString *)key;
- (BOOL)imageExistsInDisk:(NSString *)key;
- (NSUInteger)countImagesInMemory;
- (NSUInteger)countImagesInDisk;
- (void)removeImageWithKey:(NSString *)key;
- (void)removeAllImages;
- (void)removeAllImagesInMemory;
- (void)removeOldImages;

@end
```

Basically, ImageCache can be configured to have a fixed size in memory, and the images that were added first are removed first. It loads the images from the disk as required, keeping a copy in memory, and as suggested by Alex, you can remove them from memory in case of a warning:

```
- (void)applicationDidReceiveMemoryWarning:(UIApplication *)application
{
    [[ImageCache sharedImageCache] removeAllImagesInMemory];
}
```

The complete source code of this ImageCache class, together with

some unit tests (thanks to the Google Toolkit for Mac[8]), is available on the Projects section of this blog[9] for you to download and play with.

Build Custom Table Cells and Reuse Them Properly

Remember to always use static NSString identifiers for your cells, which helps the UITableView class to reuse them and reduce memory consumption:

```
- (UITableViewCell *)tableView:(UITableView *)tableView
         cellForRowAtIndexPath:(NSIndexPath *)indexPath
{
    Item *item = [items objectAtIndex:indexPath.row];
    static NSString *identifier = @"ItemCell";

    ItemCell *cell = (ItemCell *)[tableView
          dequeueReusableCellWithIdentifier:identifier];

    if (cell == nil)
    {
        cell = [[[ItemCell alloc] initWithIdentifier:identifier]
                                                        autorelease];
    }
    cell.item = item;
    return cell;
}
```

I also avoid using NIBs when working with table cells, for performance reasons. I prefer to draw the cells through my own subclasses of UITableViewCell, themselves using overridden setters for their properties, which takes me to the next point.

Override Setters Properly

As I said above, I tend to create my own subclasses of UITableViewCell, providing a simple property through which I change the model class holding the data that the cell is supposed to show. This has the effect of changing all the values of the fields and labels to the values corresponding to those of the model instance.

To do that, I override the setters as follows; for the following class definition...

```
@interface SomeClass
{
@private
    NSArray *items;
    NSString *name;
```

[8] http://code.google.com/p/google-toolbox-for-mac/
[9] https://github.com/akosma/imagecachetest

```
    id<SomeProtocol> delegate;
}

@property (nonatomic, retain) NSArray *items;
@property (nonatomic, copy) NSString *name;
@property (nonatomic, assign) id<SomeProtocol> delegate;
@end
```

... I use the following implementation:

```
@implementation SomeClass

@synthesize items;
@synthesize name;
@synthesize delegate;

- (void)dealloc
{
    [items release];
    [name release];
    delegate = nil;
}

#pragma mark -
#pragma mark Overridden setters

- (void)setItems:(NSArray *)obj
{
    if (obj == items)
    {
        return;     // thanks Marco! (see comment #3 below)
    }
    [items release];
    items = nil;
    items = [obj retain];

    if (items != nil)
    {
        // create the internal structure of the cell
        // if not present, and change the widget values
    }
}

- (void)setName:(NSString *)obj
{
    [name release];
    name = nil;
    name = [obj copy];   // I always copy NSStrings!

    if (name != nil)
    {
```

```
        // create the internal structure of the cell
        // if not present, and change the widget values
    }
}

- (void)setDelegate:(id<SomeProtocol>)obj
{
    // do not retain! This is an "assign" property
    delegate = obj;

    if (delegate != nil)
    {
        // create the internal structure of the cell
        // if not present, and change the widget values
    }
}

@end
```

Overriding setters properly is important because you might be introducing memory leaks if done wrong. I usually copy all my NSString properties too, as a rule of thumb.

Beware of Delegation

If your code is delegate of some other object which you are about to release, remember to set its delegate property to nil before releasing it; otherwise, the object might "think" that its delegate is still there, and will send a message to an invalid pointer. To see what I'm talking about, consider this code:

```
@interface SomeClass <WidgetDelegate>
{
@private
    Widget *widget;
}
@end

@implementation SomeClass

- (id)init
{
    if (id = [super init])
    {
        widget = [[Widget alloc] init];
        widget.delegate = self;
    }
    return self;
}

- (void)dealloc
```

```
{
    // widget might be retained by someone else!
    widget.delegate = nil;
    [widget release];
    [super dealloc];
}

#pragma mark -
#pragma mark WidgetDelegate methods

- (void)widget:(Widget *)obj callsItsDelegate:(BOOL)value
{
    // and here something happens...
}

@end
```

SomeClass is delegate of Widget. Widget instances might be retained by someone else, which means that even after the release message in the dealloc method, widget might still be alive and call its delegate; if this variable is not nil, widget will send a message to a non-existent object, which will surely crash your application.

Use Instruments

The "Leaks" instrument is your friend, and you should use it after you write the first line of code. Typically, I launch it every time before doing a checkin of some new code. In Xcode, select "Run / Start with Performance Tool / Leaks" and you're done. You can use it in the simulator or on your device.

Use a Static Analysis Tool

Use the LLVM/Clang Static Analyzer[10] tool. This amazing tool will catch naming errors (regarding the Objective-C naming conventions) and some hidden memory leaks, which are particularly nasty when using CoreFoundation libraries (Address Book, sound, CoreGraphics, etc). You can add it to your daily build script, it's very easy to use.

But you must use it. Enough said.

Use NSZombieEnabled

Lou Franco has posted an excellent article about how to use NSZombieEnabled[11] in your development cycle. The idea is to be able to find which messages are being sent to invalid pointers, referencing objects which have been released somewhere in your code. Always

[10] http://clang.llvm.org/StaticAnalysis.html
[11] http://loufranco.com/blog/files/debugging-memory-iphone.html

remember who's the owner of your objects, and check for existence elsewhere!

And You?

How about you? What are your tips or best practices you usually use for your iPhone apps? Feel free to share them in the form below.

Update, 2009-01-29: I am overwhelmed with the response and traffic that this post has gotten so far! Yesterday evening I had the pleasure of discussing this subject with the guys of the iPhone Developers Facebook group, and I got interesting remarks from Marco Scheurer from Sen:te[12] (including a comment below), which I've added to this post today.

Oh, and by the way, I've uploaded the slides here![13] They have a Creative Commons license, so feel free to use them if you find them useful.

[12] http://www.sente.ch/

[13] 10_Tips_Memory_Presentation.pdf.zip

Mention on iPhoneFlow

Adrian Kosmaczewski

2009-02-12

Published in iPhoneFlow[1].

> Adrian Kosmaczewski presents 10 code-driven iPhone memory management tips. A really good read!

[1] http://www.iphoneflow.com/items/1602

Je Vends Mon Vieux G5

Adrian Kosmaczewski

2009-02-14

Pour ceux qui seraient intéressés, je vends mon Power Mac G5 (acheté 2003) et son écran, plus toute une série d'accessoires, sur Anibis.ch[1] et sur Ricardo.ch[2].

Ordinateur de bureau Apple PowerMac (Novembre 2003) avec une série de bonus hors du commun; une affaire à ne pas louper!

- Nom du modèle: Power Mac G5
- Identifiant du modèle: PowerMac7,2
- Nom du processeur: PowerPC 970 (2.2)
- Vitesse du processeur: 2 GHz
- Nombre de processeurs: 2
- Cache de niveau 2 (par processeur): 512 Ko
- Mémoire: 1 Go
- Vitesse du bus: 1 GHz
- Version de la ROM de démarrage: 5.1.3f0
- Numéro de série: CK34805QNVB
- 1 port Firewire 800
- 2 ports Firewire 400 (dont 1 frontal)
- 3 ports USB 2.0 (dont 1 frontal)
- Sortie video DVI
- Entree/Sortie audio numérique
- Port Ethernet
- Bluetooth
- Carte audio + speakers intégrés
- Modem intégré
- Disque dur de 233.76 GB Serial ATA (Maxtor 6Y250M0)
- Graveur CDÂ±RW & DVD-RW (PIONEER DVD-RW DVR-106D; peut graver et lire des CD-R, CD-RW, DVD-R, DVD-RW, DVD+R, DVD+RW)
- 3 "full-length" PCI-X slots: un a 133 MHz, 64-bit slot et deux 100 MHz, 64-bit slots.
- Carte Graphique ATI Radeon 9700 Pro (VRAM totale: 64 Mo)
- Ecran Apple Cinema Display 20'' (1680 x 1050)

Accessoires inclus:

[1] http://www.anibis.ch/n/2280600

[2] http://www.fr.ricardo.ch/acheter/ordinateurs-et-reseaux/apple/power-mac/powermac-g5-2x-2ghz--cinema-display--lot-d-accessoires/v/an562115822/

- Borne Airport Extreme (2005) avec accessoires;
- Caméra iSight (externe);
- Adaptateur externe DVI vers VGA;
- DVD Mac OS X Leopard Family Pack (5 licenses) préinstallé, avec les dernieres mises a jour (10.5.6);
- DVD iLife ’08 (aussi préinstallé)
- 2 Clavier Suisses, un sans-fil et un autre “filaire” (aussi modèles 2003);
- 2 Souris (“filaires”) Apple “Mighty Mouse”;
- Router ADSL Linksys + cables et filtre;
- Switch Ethernet marque Roline (4 ports);
- Cables divers.

En parfait etat de fonctionnement (sans son emballage d’origine). Voir photos incluses! (sur Anibis.ch[3] ou Ricardo.ch[4]. ;)

Voici les specs complètes de l’ordinateur[5] et de l’ecran[6].

Estimations de prix d’apres everymac.com (mises a jour le 10 février 2009):

- Power Mac G5 2 x 2GHz: USD 750 a USD 950 (~850 CHF a ~1100 CHF)
- Cinema Display 20’’: USD 599 (~700 CHF)

Update, 2009-02-21: Vendu! Ricardo.ch[7] a rempli son rôle parfaitement.

[3] http://www.anibis.ch/n/2280600

[4] http://www.fr.ricardo.ch/acheter/ordinateurs-et-reseaux/apple/power-mac/powermac-g5-2x-2ghz--cinema-display--lot-d-accessoires/v/an562115822/

[5] http://www.everymac.com/systems/apple/powermac_g5/stats/powermac_g5_2.0_dp.html

[6] http://www.everymac.com/monitors/apple/studio_cinema/specs/apple_cinema_display_20_2.html

[7] http://www.fr.ricardo.ch/

Olé, olé, olé

Adrian Kosmaczewski

2009-02-15

I just stumbled into this amazing TED talk by Elizabeth Gilbert[1] via James Duncan Davidson[2] (@duncan[3] in Twitter) and I want to share it with you with some very personal thoughts below.

I've been fortunate enough to earn a pretty decent living doing basically what I consider a hobby for the past 13 years, which is typing code on a computer and see if it works. Which most of the time doesn't, but that's part of the game.

I strongly believe in what Elizabeth says in this talk, and I have believed in this for years. I deeply believe that we, software developers, software engineers, both self-taught and those coming out of college, are just creators, just as Elizabeth describes them. Simple creators, being able to provide new ways to information to be shown, to flow, to entertain, to move. Simple channels through which ideas are transformed into tools, behaviours, images and sound.

There's been a huge debate in this matter. Knuth named his masterpiece "The Art of Computer Programming"[4], and the single choice of this title has sparkled a longlasting debate in the software community, one that this essay is unfortunately going to feed, too.

Interestingly enough, Knuth developed his own typesetting system for his book, TeX[5], which is named after the Greek word meaning "art or craft". His work not only had to be the most important book ever written on programming, but also, it had to be beautiful.

It had to be an object of art.

And I think that programming itself is art. And I think that programmers are artists. And this is maybe the single reason why so much has been written about programmer productivity, why software project management is so hard, why discussions around programming languages distort into trolls and heated arguments, and why you feel this anger against this words on my blog and you call me names.

[1] http://www.elizabethgilbert.com/

[2] http://duncandavidson.com/2009/02/one-of-my-favorite-ted2009-tal.html

[3] http://twitter.com/duncan

[4] http://en.wikipedia.org/wiki/The_Art_of_Computer_Programming

[5] http://en.wikipedia.org/wiki/TeX

This is why writing opinionated software[6] is key to success, that's why the best software companies take time into creating great software development environments that stand out[7], that's why Peopleware[8] is so important, even 20 years after being published for the first time.

It is all about letting the flow of art come through the person whose hands are on the keyboard. It's all about letting this happen. It is not us who write, it is the writing that comes to us.

Software is art, and as such, it needs time, patience, iterations, silence, passion, coffee, naps, pizza, books, compilers, laughs, Nintendo Wiis and unit testing suites.

The creation of good software is embodied in the creative process itself. The best engineers I know suffer from this process as much as they enjoy it, using an iterative process of trial and error which, even after all these years, still applies. The best software developers release sometimes early, sometimes late, sometimes with quality, sometimes not, but they release. They refactor. They document. They teach others about all of this. They always think that they can do better.

They always think, as Elizabeth says in her talk, that their current work is the worst in the history of programming: "Not just bad, but the worst". They suffer about it. But they release, and they fight against the fear of being critisized because of their choice of programming languages, operating systems, tools, processes, insufficient testing or design patterns.

Real artists ship.[9] The making of this industry is full of examples of why software is an art: the first Macintosh, Smalltalk, NeXTstep, the Internet, Erlang, Apache, Ruby on Rails, UNIX & C, Lisp; just glimpses of wisdom, brilliantly crafted, that struck as obvious yet incredible, and which prompt a huge crowd to cheer up and applause.

Anyway, I'm not as famous or well-known as Elizabeth or Duncan. I have not yet done anything such as Ant[10] or Tomcat[11] (originally written by Duncan, by the way), even if I release software and projects with an increasingly high rate lately, and with many projects in the pipeline these days. I hope that my best successes are still ahead of me. And I hope you'll enjoy them one day, too.

Do I think that a little "genie" is besides me? Yes I do. And given the extremely rational background of most engineers out there, stating such an argument will raise more chuckles than anything else. Heck, who cares.

---

[6] http://gettingreal.37signals.com/ch04_Make_Opinionated_Software.php

[7] http://www.google.com/support/jobs/bin/static.py?page=about.html

[8] http://www.amazon.com/Peopleware-Productive-Projects-Teams-Second/dp/0932633439

[9] http://www.folklore.org/StoryView.py?story=Real_Artists_Ship.txt

[10] http://ant.apache.org/

[11] http://tomcat.apache.org/

If you ask me, there is something magic out there.

PS: there's this quote attributed to Jorge Luis Borges[12] which says that "publishing is a way to stop editing"... and I thought about it just after publishing this post. I don't know if he really said that, but in any case I agree. The difference being that, in our case, we refer to publishing as "releasing". But the feeling is the same.

PS (2): I've already written[13] about the importance of delivering working software. I just forget about all I've written or linked to[14].

[12] http://en.wikipedia.org/wiki/Jorge_Luis_Borges
[13] /2007/11/11/deliver-now/
[14] /2007/05/25/15-startup-commandments-by-mark-fletcher/

Going Github

Adrian Kosmaczewski

2009-02-18

This is something I've been looking forward to do for some time. After praising git back in 2007, now I'm moving many of my personal projects to Github, which has an absolutely brilliant service! For the moment I'm using the free account, but I will most probably switch to a paid account soon. The only thing it lacks, in my opinion, is a bug & issue tracker as you have in Google Code repositories, but other than that, it's simply perfect.

So feel free to check out the projects I've moved there, and of course, to fork them and enjoy the code as you see fit.

Mention on StackOverflow

Adrian Kosmaczewski

2009-03-10

Quote appeared in Stack Overflow[1].

> Hope Adrian Kosmaczewski's work can save your time from reinventing the wheels: http://github.com/akosma/iphone restwrapper/tree/master And, it's Public Domain.

[1] http://stackoverflow.com/questions/630306/iphone-rest-client

Quotes

Adrian Kosmaczewski

2009-03-12

A small compilation of quotes I've put below the header of this blog during the past few years:

- No vemos las cosas como son. Las vemos como somos. (Hilario Ascasubi[1]) - We don't see things as they are; we see them as we are.
- Don't be afraid to try something new. An amateur built the ark. Professionals built the Titanic. (unknown)
- Compatibility means deliberately repeating other people's mistakes. (David Wheeler[2])
- Cuando uno se compromete con lo que piensa, y encima piensa cosas que cuestionan lo incuestionable, es de esperar que haya alguna dificultad. (hernún[3]) - When you stick to your ideas, and on top of that you think about questioning what's out of question, it's likely there'd be some problems.
- The definition of insanity is doing the same thing over and over and expecting different results. (Benjamin Franklin)
- Most people are fools, most authority is malignant, God does not exist, and everything is wrong. (Ted Nelson[4])
- sin incertidumbre no hay novedad, sin novedad posible no hay más que repetición y, por lo tanto, negación del otro como un ser libre: el ser libre es un ser incierto. (adrian mancuso) - without uncertainty there's no novelty, without novelty there's only repetition and, therefore, negation of the other as a free being: being free is being unpredictable.
- Software is like entropy. It is difficult to grasp, weighs nothing, and obeys the second law of thermodynamics; i.e. it always increases (Norman R. Augustine)

[1] http://en.wikipedia.org/wiki/Hilario_Ascasubi
[2] http://www.dwheeler.com/
[3] http://hernun.com.ar/
[4] http://www.wired.com/wired/archive//3.06/xanadu_pr.html

NIBs or code? Why not both? Here's nib2objc.

Adrian Kosmaczewski

2009-03-17

(Somehow this project seems to me so simple, that I'm sure someone has done this before. Anyway). This is my feeble attempt to bring an answer to the eternal dichotomy between those arguing about the relative benefits of creating user interfaces via Interface Builder or via pure Objective-C code: let me introduce nib2objc[1].

Unbeknown to most of us, the ibtool[2] utility bundled with Interface Builder and Xcode allows us to inspect the contents of NIB files (or XIBs, for that matter) and get from them nice property lists XML streams, which I transform in NSDictionary instances, which I loop over and over util I get something that looks like this:

```
UIView *view6 = [[UIView alloc] initWithFrame:CGRectMake(0.0, 0.0, 320.0, 460.0
view6.frame = CGRectMake(0.0, 0.0, 320.0, 460.0);
view6.alpha = 1.000;
view6.autoresizingMask = UIViewAutoresizingFlexibleWidth | UIViewAutoresizingFl
view6.backgroundColor = [UIColor colorWithWhite:0.750 alpha:1.000];
view6.clearsContextBeforeDrawing = NO;
// ...

UIButton *view9 = [UIButton buttonWithType:UIButtonTypeRoundedRect];
view9.frame = CGRectMake(167.0, 65.0, 72.0, 37.0);
view9.adjustsImageWhenDisabled = YES;
view9.adjustsImageWhenHighlighted = YES;
view9.alpha = 1.000;
view9.autoresizingMask = UIViewAutoresizingFlexibleRightMargin | UIViewAutoresi
view9.clearsContextBeforeDrawing = NO;
view9.clipsToBounds = NO;
view9.contentHorizontalAlignment = UIControlContentHorizontalAlignmentCenter;
// ...
[view9 setTitleShadowColor:[UIColor colorWithWhite:0.000 alpha:1.000] forState

// ...
```

[1] http://github.com/akosma/nib2objc/

[2] http://developer.apple.com/DOCUMENTATION/DARWIN/Reference/ManPages/man1/ibtool.1.html

```
[view6 addSubview:view9];
// ...
```

Using this tool, I can now use IB for design, and then generate the code for those designs, in case I prefer to use a code-only approach (usually for UITableViewCells, as I explained before[3]). For the moment it only works with UIKit classes, but I don't think it might be a problem to extend it to AppKit classes as well.

I hope this project is useful to all of you! As usual, open source, public domain and on Github[4].

Update, 2009-04-09: This project has been featured in an article in Ars Technica[5] by Erica Sadun[6]!

[3] /2009/01/28/10-iphone-memory-management-tips/

[4] http://github.com/akosma/nib2objc/

[5] http://arstechnica.com/apple/guides/2009/04/iphone-dev-convert-xib-files-to-objective-c.ars

[6] http://ericasadun.com/

nib2objc

Adrian Kosmaczewski

2009-03-17

(Somehow this project seems to me so simple, that I'm sure someone has done this before. Anyway). This is my feeble attempt to bring an answer to the eternal dichotomy between those arguing about the relative benefits of creating user interfaces via Interface Builder or via pure Objective-C code: let me introduce nib2objc[1].

Unbeknown to most of us, the ibtool[2] utility bundled with Interface Builder and Xcode allows us to inspect the contents of NIB files (or XIBs, for that matter) and get from them nice property lists XML streams, which I transform in NSDictionary instances, which I loop over and over util I get something that looks like this:

```
UIView *view6 = [[UIView alloc] initWithFrame:CGRectMake(0.0, 0.0, 320.0, 460.0
view6.frame = CGRectMake(0.0, 0.0, 320.0, 460.0);
view6.alpha = 1.000;
view6.autoresizingMask = UIViewAutoresizingFlexibleWidth | UIViewAutoresizingFl
view6.backgroundColor = [UIColor colorWithWhite:0.750 alpha:1.000];
view6.clearsContextBeforeDrawing = NO;
// ...

UIButton *view9 = [UIButton buttonWithType:UIButtonTypeRoundedRect];
view9.frame = CGRectMake(167.0, 65.0, 72.0, 37.0);
view9.adjustsImageWhenDisabled = YES;
view9.adjustsImageWhenHighlighted = YES;
view9.alpha = 1.000;
view9.autoresizingMask = UIViewAutoresizingFlexibleRightMargin | UIViewAutoresi
view9.clearsContextBeforeDrawing = NO;
view9.clipsToBounds = NO;
view9.contentHorizontalAlignment = UIControlContentHorizontalAlignmentCenter;
// ...
[view9 setTitleShadowColor:[UIColor colorWithWhite:0.000 alpha:1.000] forState

// ...
[view6 addSubview:view9];
// ...
```

[1] http://github.com/akosma/nib2objc/

[2] https://web.archive.org/web/20090326094828/http://developer.apple.com/DOCUMENTATION/DARWIN/Reference/ManPages/man1/ibtool.1.html

Using this tool, I can now use IB for design, and then generate the code for those designs, in case I prefer to use a code-only approach (usually for UITableViewCells, as I explained before[3]). For the moment it only works with UIKit classes, but I don't think it might be a problem to extend it to AppKit classes as well.

I hope this project is useful to all of you! As usual, open source, public domain and on Github[4].

Update, 2009-04-09: This project has been featured in an article in Ars Technica[5] by Erica Sadun[6]!

Update, 2010-07-18: Major update to the project! Check out the latest features[7] and the new versions[8] (including a Mac OS X GUI application!)

[3] /blog/10-iphone-memory-management-tips/

[4] http://github.com/akosma/nib2objc/

[5] http://arstechnica.com/apple/guides/2009/04/iphone-dev-convert-xib-files-to-objective-c.ars

[6] http://ericasadun.com/

[7] /blog/nib2objc-updated/

[8] /blog/more-nib2objc-fun/

iPhone SDK 3.0: A New Beginning

Adrian Kosmaczewski

2009-03-22

Last year I blogged[1] about the upcoming SDK 2.0 for the iPhone 3G, and boy did it change my life. For those who haven't followed closely everything that happened in this blog lately, there's been this[2] (that's me in the WWDC keynote main room at the Moscone center) and then that[3] (yours truly talking at the first ever European iPhone conference). All of this has been the result of going to San Francisco last June. That particular trip changed everything; I never thought that a simple plane ticket could generate this much.

The iPhone has literally changed my professional life. But it was only the beginning. Last Tuesday, Apple announced the iPhone SDK 3.0[4], and I'll expose here some thoughts about what's coming next.

To put it bluntly, I think that the iPhone SDK 3.0 is a jump to the realm of the desktop software platform. **By that I mean that the next generation of iPhone applications will look more like small versions of more complex desktop-like systems, rather than mobile applications.** Pasteboard, local Bluetooth networking, Undo support and Core Data are just some of the elements that will take the iPhone platform to the level of a small desktop platform (and no, I'm not breaking any NDA here, I'm just enumerating some of the "1000 new APIs" announced by Apple last week).

[1] /blog/iphone-sdk-une-nouvelle-ere-démarre/
[2] /blog/i-was-there/
[3] /blog/iphone-conference-2008-a-bit-of-magic/
[4] http://events.apple.com.edgesuite.net/0903lajkszg/event/index.html

Much has been written about the App Store review process, about the lack of X, Y or Z features on the SDK, about the various problems and limitations of the platform; however, constraints are liberating[5]. All of the limitations of the iPhone SDK have allowed lots of developers to come up with creative ideas to overcome them, and to create applications with a seriously distinctive taste to them; we wouldn't have had Ocarina or Shazam or the Facebook iPhone app otherwise.

Have there been a similar breakthrough in the Google Android platform? Let's be very clear about this: **the answer is no**. I've personally seen the Android SDK in action, and even if the Android platform seems promising, Apple's own SDK is years light ahead. As an advocate of open source solutions, I am sorry to say this, but Android is not a direct competitor of the iPhone SDK, at least not right now.

And not only that, but there's one important factor that's generally overseen: the strict approval factor for getting into the App Store has generated one of the most secure and stable software platforms ever delivered to the public. Have you heard about iPhone viruses? You haven't, right? Well, I've heard about Windows Mobile ones[6], if you ask me. And you can buy antivirus software for Blackberry and Android[7], by the way.

Heck, the first iPhone "virus" was the 1.1.3 firmware[8], it was released by Apple… and it screwed jailbroken iPhones. Big deal.

The platform might not be perfect, but hey, I can't think of working on any other right now.

And then, there's the programming language choice; as a geek and developer I can't think of a nicer choice than Objective-C's dynamicity and speed to deliver great software. Java-based platforms are limited by, well, Java itself[9]. Want speed? Use straight C. Want object-orientation? Use Objective-C. Want compatibility and reuse? Mix C++ with your Objective-C code. Let's go back to the basics: the iPhone is delivering what no other mobile platform has dared doing before.

And, oh, by the way, don't rant about not having background processes (or any other feature, for that matter); I really don't think that the lack of support for them will hinder the development of this platform. Rather the opposite, we'll see a new generation of applications popping up in June, when the OS 3.0 will be released to the public. I'm more confortable with a platform with defined boundaries, rather than with a behemoth of unknown and undefined behaviour. I'll be glad to use the notification service.

The iPhone SDK might be limited, but it delivers what it promises. I

[5] http://gettingreal.37signals.com/ch03_Embrace_Constraints.php

[6] http://www.informit.com/articles/article.aspx?p=337069

[7] http://secure.smobilesystems.com/

[8] http://iphonetouch.blorge.com/2008/01/08/worlds-first-iphone-virus-sucks-as-a-virus/

[9] http://twitter.com/nst021/statuses/1244270786

have yet to see a platform from another vendor (or even from the open source realm) to be as coherent and as well defined from the very start.

The real power of the iPhone remains yet to be seen. This platform is beyond anything that we've seen so far, and all the elements for success are already present in the beta downloads.

To summarize: the iPhone is the next desktop platform.

But what about Google Android, Symbian or Blackberry? Of course, they are interesting alternatives. The iPhone is by no means the right answer for everyone, but there's no doubt to me that Apple is setting the standard, here and now. Apple is leading the way, and iPhone shows what mobile smartphones should have been (and done) from the very beginning.

Please stay tuned for more goodies. You ain't see nothin' yet.

Sooshi

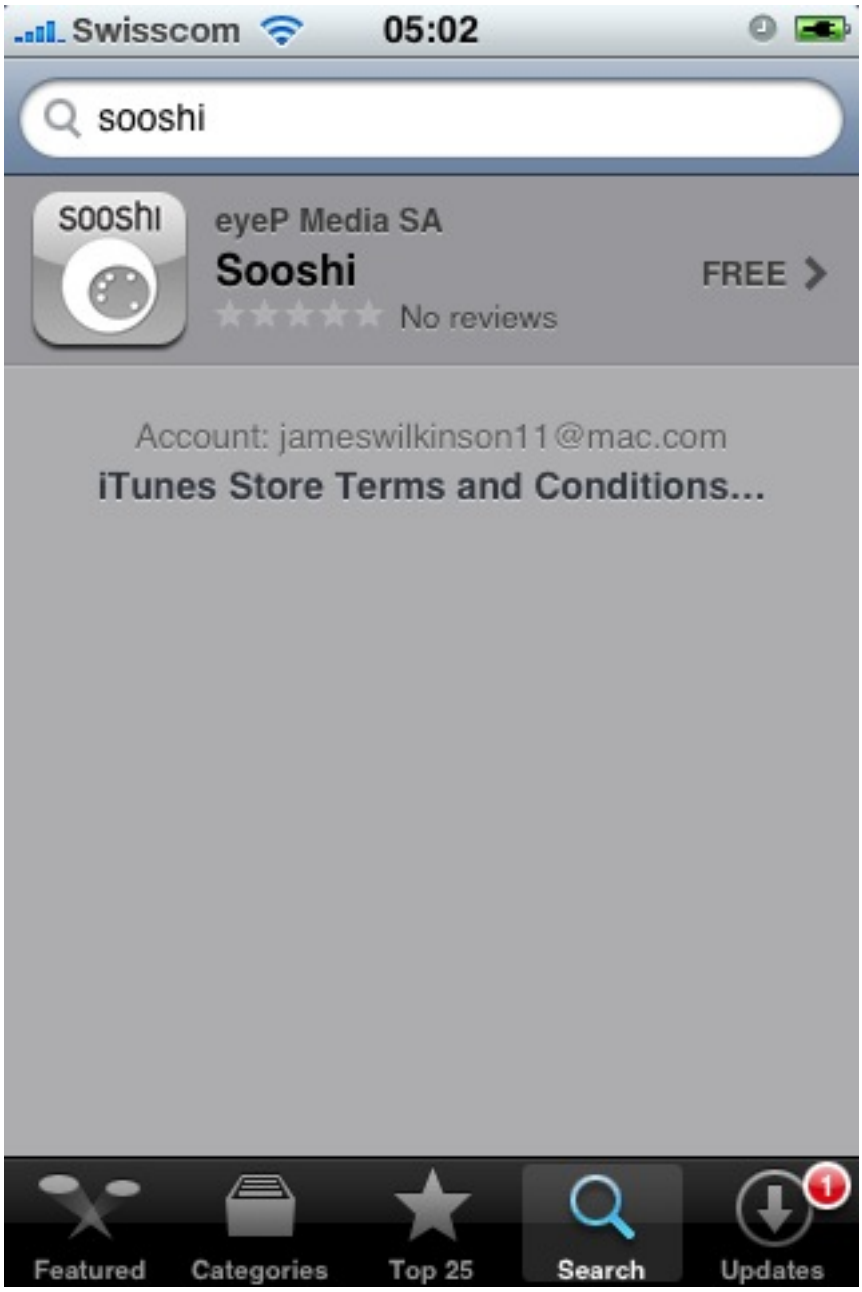
Adrian Kosmaczewski

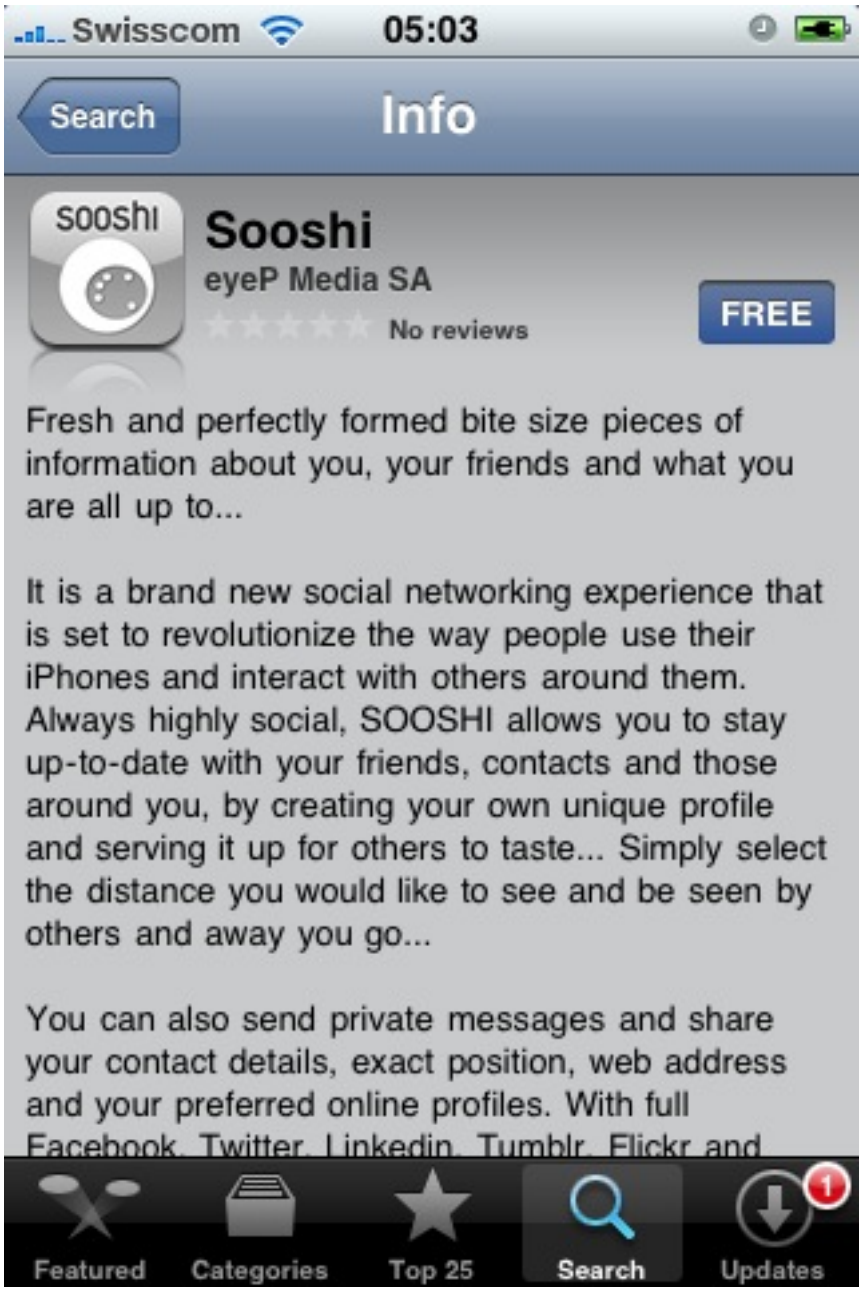
2009-03-26

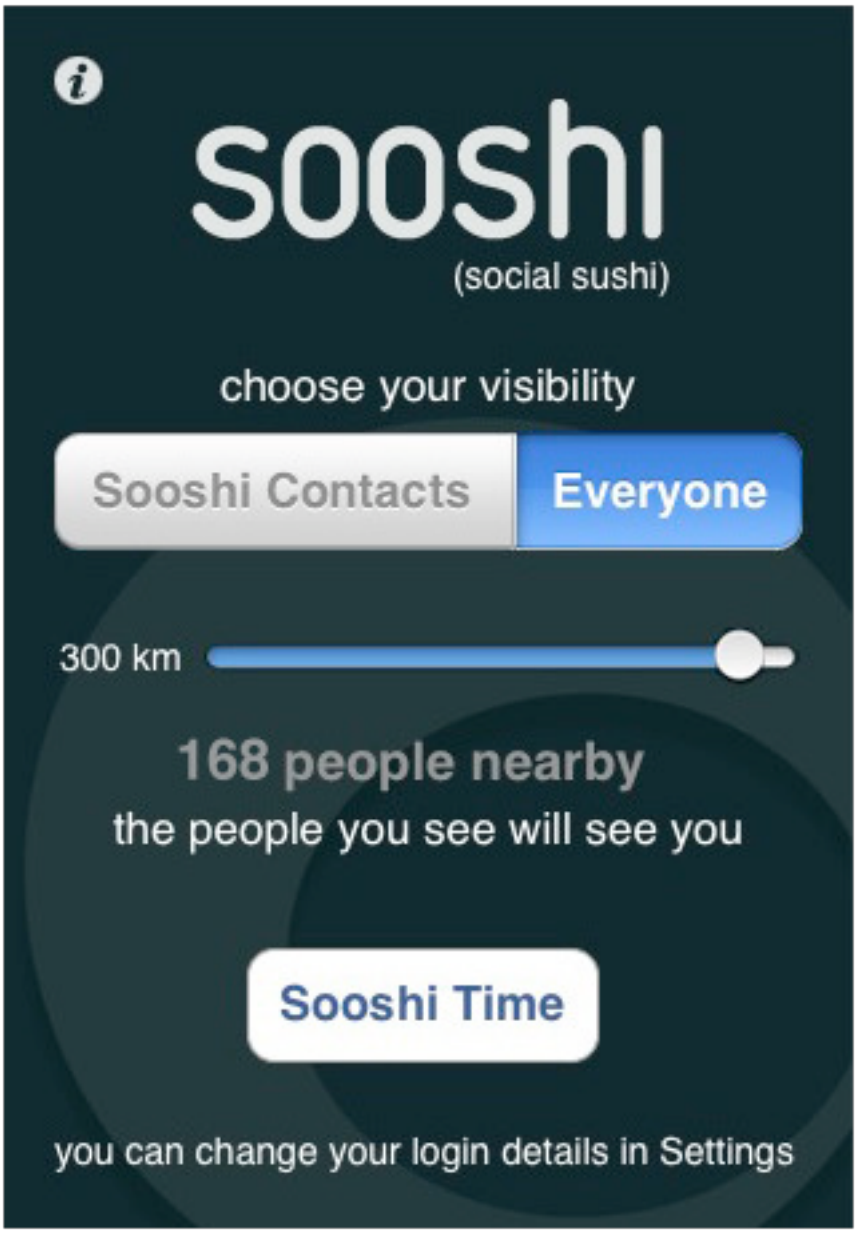
This is the big, secret iPhone application project I've been working on for the past 9 months. Sooshi is out in an App Store near you.[1] My name isn't on it, but I am responsible of 99% of its code, both the client and the server behind it. It's like my baby, but given for adoption.

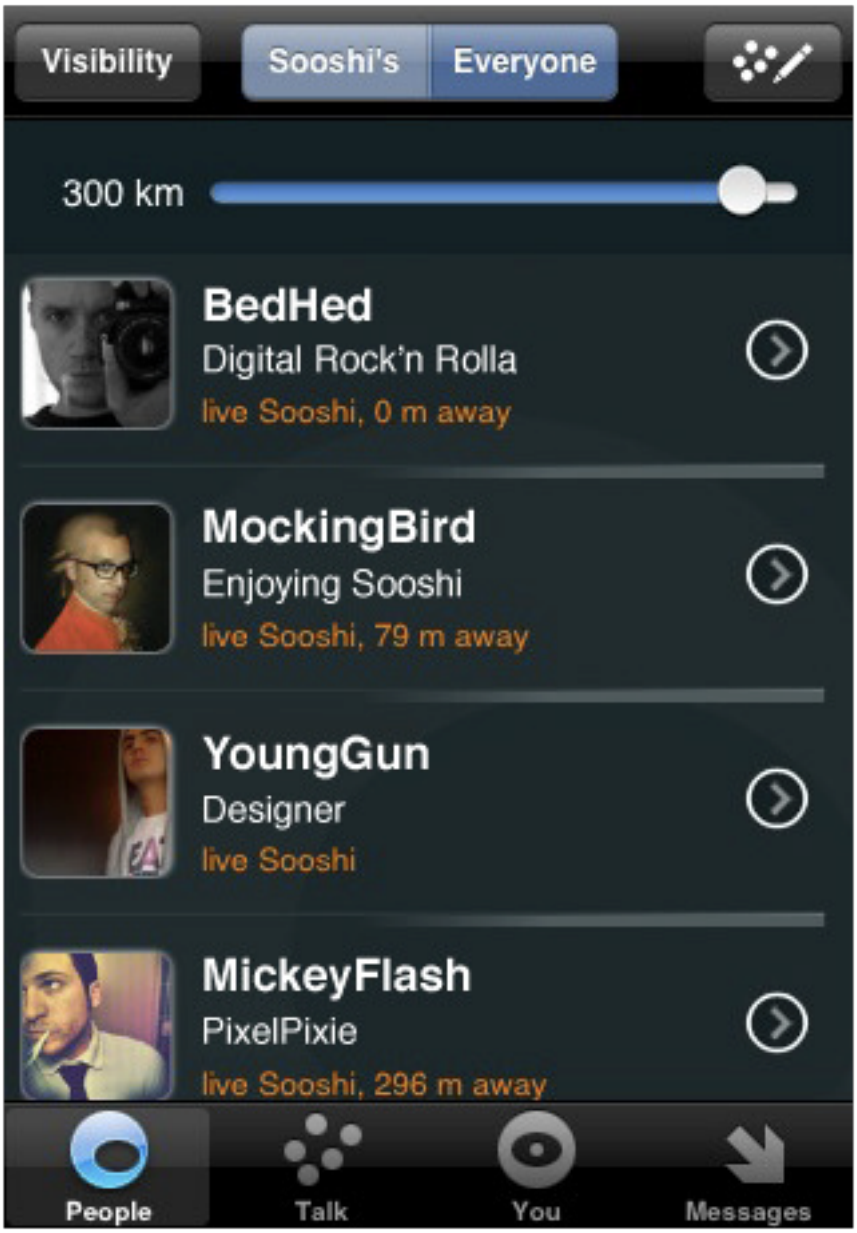
The design is James', also known as John Smith ;) of course, he's the one in charge to make things pretty :) I just try to make them work properly, that's all.

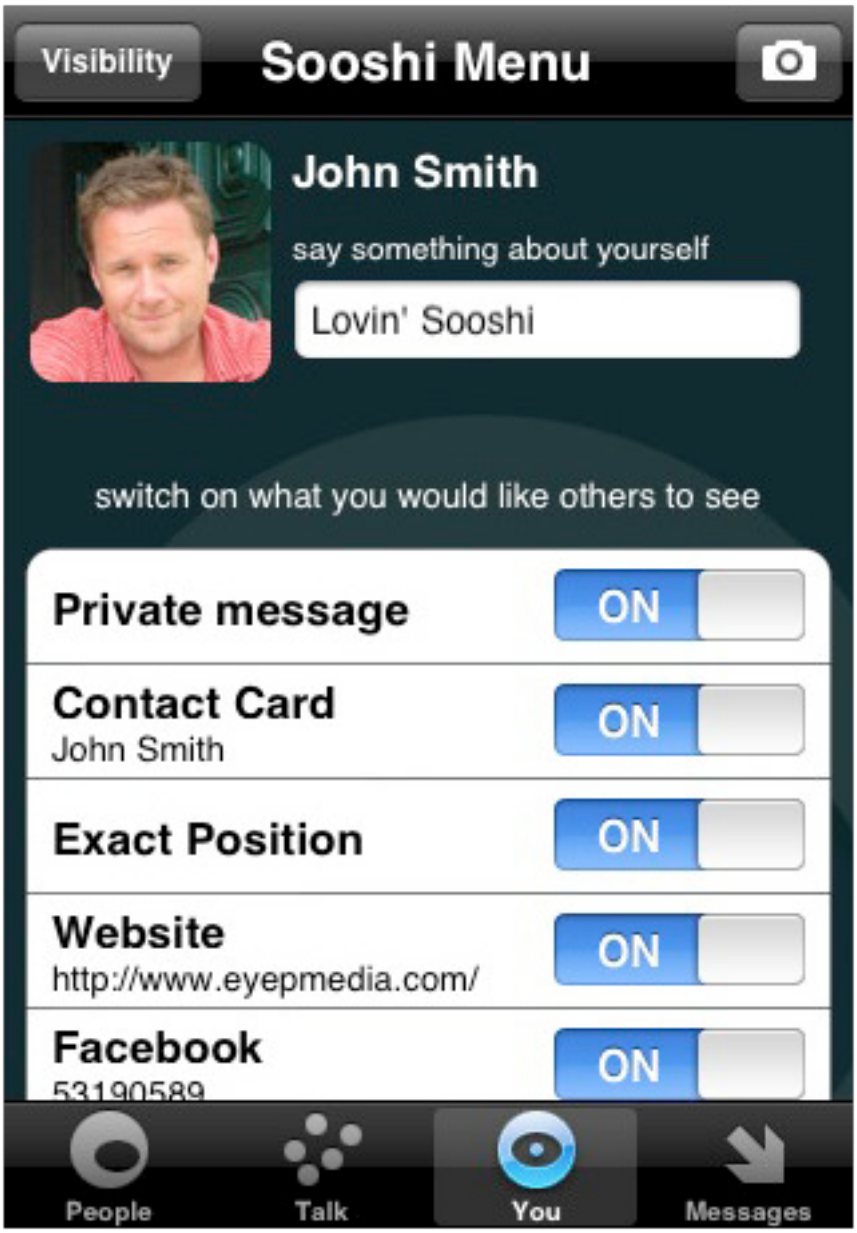
[1] http://itunes.apple.com/WebObjects/MZStore.woa/wa/viewSoftware?id=305714327

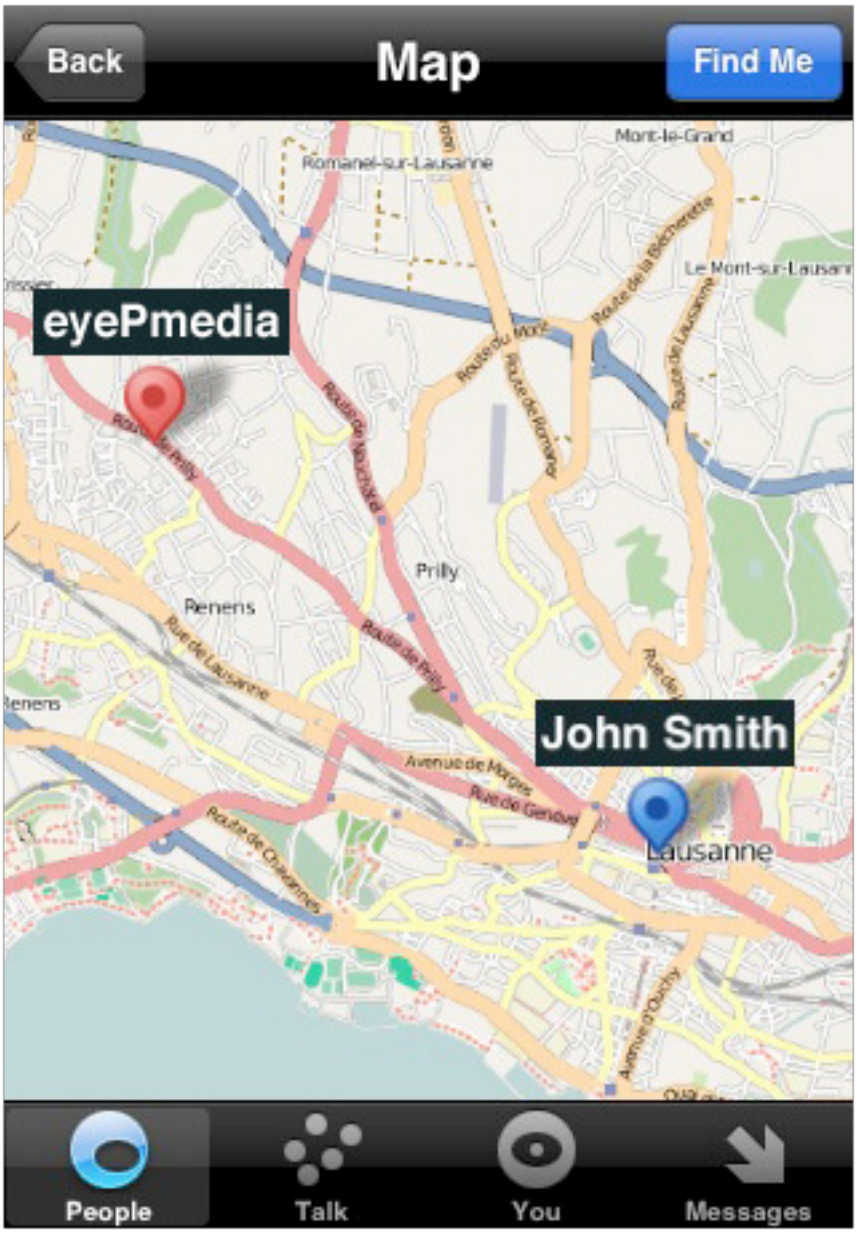

Swisscom
05:02
sooshi
sooshi
eyeP Media SA
Sooshi
FREE
No reviews
Account: jameswilkinson11@mac.com
iTunes Store Terms and Conditions...
Featured
Categories
Top 25
Search
Updates
1


Swisscom
05:03
Search
Info
Sooshi
Sooshi
eyeP Media SA
No reviews
FREE
Fresh and perfectly formed bite size pieces of information about you, your friends and what you are all up to...
It is a brand new social networking experience that is set to revolutionize the way people use their iPhones and interact with others around them. Always highly social, SOOSHI allows you to stay up-to-date with your friends, contacts and those around you, by creating your own unique profile and serving it up for others to taste... Simply select the distance you would like to see and be seen by others and away you go...
You can also send private messages and share your contact details, exact position, web address and your preferred online profiles. With full
Featured
Categories
Top 25
Search
Updates


sooshi
(social sushi)
choose your visibility
Sooshi Contacts
Everyone
300 km
168 people nearby
the people you see will see you
Sooshi Time
you can change your login details in Settings


Visibility
Sooshi's
Everyone
300 km
BedHed
Digital Rock'n Rolla
live Sooshi, 0 m away
MockingBird
Enjoying Sooshi
live Sooshi, 79 m away
YoungGun
Designer
live Sooshi
MickeyFlash
PixelPixie
live Sooshi, 296 m away
People
Talk
You
Messages


Visibility
Sooshi Menu
John Smith
say something about yourself
Lovin' Sooshi
switch on what you would like others to see
Private message
ON
Contact Card
John Smith
ON
Exact Position
ON
Website
http://www.eyepmedia.com/
ON
Facebook
53190589
ON
People
Talk
You
Messages



Enjoy! I hope you like it and don't hesitate to give us your feedback on it. We're already working in the future versions of Sooshi (plus other cool projects waiting for approval)! Stay tuned for more App Store happiness.

600th Post and Mobile Version

Adrian Kosmaczewski

2009-03-29

This is the 600th post in this blog! It all started with this post[1] written in the airport of Buenos Aires, coming back to Switzerland. And the news of the day is that, thanks to Donncha O Caoimh[2], the new WordPress Super Cache[3] plugin now works with MobilePress[4], which officially enables a mobile version of this site.

Thanks to all of you who visit this blog (you're over 6'000 every month!), to all of you who've left 563 comments (no mention of the 28,304 spam comments caught by Akismet[5]) and I hope that my writing will still be of interest to all of you in the years to come.

[1] /2004/11/06/justo-antes-de-irme/
[2] http://ocaoimh.ie/
[3] http://ocaoimh.ie/wp-super-cache/
[4] http://mobilepress.co.za/
[5] http://akismet.com/

Mention on ArsTechnica

Adrian Kosmaczewski

2009-04-07

Article published by ArsTechnica[1].

> A handy open source utility by Adrian Kosmaczewski allows you to convert Interface Builder files to Objective-C code. With it, you can extract all the layout information and properties of your visual design and transform that into code. Nib2objc does exactly what its name suggests. With it, you can generate converted code that takes into account the class constructors, method calls, and more.

[1] http://arstechnica.com/apple/guides/2009/04/iphone-dev-convert-xib-files-to-objective-c.ars

Mention on Matt Gallagher's Blog

Adrian Kosmaczewski

2009-04-27

Published on Matt Gallagher's Cocoa With Love[1].

> Layout of the content in code is probably the weakest part of the approach I've presented. To make it easier, you can pre-layout everything in Interface Builder and copy the layout into code. For complicated layouts, you could even try using nib2objc to convert your XIB files to code automatically (although I've never done this, I'm just mentioning nib2objc because the idea is so cool).

[1] http://cocoawithlove.com/2009/04/easy-custom-uitableview-drawing.html

Mention on Mauro Del Rio's Blog

Adrian Kosmaczewski

2009-04-30

Published on Mauro Del Rio's blog[1].

> I just read an interesting post on Cocoa with Love, and I discover an interesting tool to make my life easier: Now I can create my UI with IB, and the tool will convert it to a .m file. GREAT!!! The tool is called nib2objc and it can be found on Github.

[1] http://maurodelrio.com/2009/05/01/learning-iphone-ui-application/

Another Mention on StackOverflow

Adrian Kosmaczewski

2009-05-03

> I’ve used Adrian Kosmaczewski’s iPhone Rest Wrapper with success.

Published on Stack Overflow[1].

[1] http://stackoverflow.com/questions/821967/code-for-syncing-the-iphone-with-rest-servers

WWDC 2009

Adrian Kosmaczewski

2009-06-07

I arrived to San Francisco yesterday night, after a dreadful connection in Frankfurt (note to self: to never, ever again book connecting flights with less than 90 minutes in between) and a great flight across Greenland and Canada. My internal clock insists in saying that I had breakfast at 5 o'clock, but other than that, I feel great, really excited!

That flight, by the way, could have been dubbed the "WWDC Express", as the number of guys (and gals!) typing code on Xcode was waaaaaay above the average. I had the opportunity to chat with Markus Palmanto from Finland and he showed me his amazing Accordio application! Check it out on the App Store. One of the best music instruments I've seen so far on the iPhone - from a great musician, too!

I won't go through all the fuss and rumors about the next version of the iPhone... but indeed, I'm sure that we're ready for big surprises tomorrow; for the moment, I know that tonight I'll be attending sfMacIndie, and tomorrow evening the iPhone Intelligence party; I hope to meet many of you this week!

WWDC 2009: a message for Scott Forstall

Adrian Kosmaczewski

2009-06-08

I won't go into details into all the stuff shown during the keynote[1]; this is just a single comment for Scott Forstall[2]: **STOP THE BLOODY DEMOS.** Last year it was pretty unbearable, yet this year you managed to make it even worse.

Thankfully I am not the only one[3] who thinks that the interminable[4] series of demos is just a waste of our time, when all we want is to see new stuff in the SDK[5]. We're developers[6], not marketing people.

[1] http://blog.duncandavidson.com/2009/06/wwdc-keynote-reactions.html
[2] http://en.wikipedia.org/wiki/Scott_Forstall
[3] http://twitter.com/wilshipley/status/2079295591
[4] http://twitter.com/calissendorff/status/2079381490
[5] http://twitter.com/bdudney/status/2079132642
[6] http://twitter.com/jeff_lamarche/status/2079347841

Mention sur Webd

Adrian Kosmaczewski

2009-06-09

Published in Webd[1].

> Dans ce dernier cas de figure, il existe maintenant une solution: nib2objc. Ce projet Open-Source est tout simplement un convertisseur de fichiers nib (.xib) vers du code Objectif-C. Il gère toutes les propriétés publiques de chacun des éléments graphiques, le constructeur et la hiérarchie des vues. Pour le moment, il n'y a que les composants d'UIKit qui sont supportés. L'utilisation est on-ne-peut-plus simple.

[1] http://webd.fr/735-nib2objc-ou-comment-convertir-un-nib-en-code-objectif-c

Article on the Tages Anzeiger

Adrian Kosmaczewski

2009-06-11

> Zum Umdenken hat Apple selber aktiv beigetragen, indem der Konzern das iPhone im Sommer 2008 für aussenstehende Entwickler öffnete und auf die extreme Geheimhaltung der früheren Jahre verzichtete. Seither kamen schon mehr als 50 000 Anwendungen zustande, und Hunderte von Entwicklern erzielen damit ein regelmässiges Einkommen. «Es sind diese selbstständigen Programmierer, die Apple das Sektiererische nehmen», meint Adrian Kosmaczewski, ein Entwickler aus Lausanne, der zusammen mit zwei Kollegen angereist ist und kurz davor steht, ein eigenes Unternehmen für iPhone-Anwendungen zu gründen. «Apple war wie eine königliche Familie, leicht verdorben und leicht durchgeknallt. Entwickler wie wir, die von aussen dazustossen, bringen das nötige frische Blut.»

Article by Walter Niederberger published on the Tages Anzeiger[1] (download the PDF scan of the article[2]).

[1] http://www.tagesanzeiger.ch/digital/computer/Steve-Jobs-Abwesenheit-hat-Apple-gut-getan/story/30178785

[2] /press/TagesAnzeiger-WWDC.pdf

Interview on the SonntagsZeitung

Adrian Kosmaczewski

2009-06-13

Interview by Barnaby Skinner published on the SonntagsZeitung[1] (download the PDF scan of the article[2]).

> **“Apple öffnet sich nur halb” Schweizer Entwickler berichtet von WWDC**
>
> **Das Motto der Worldwide Developer Conference von Apple lautet heuer «ein Jahr später, Lichtjahre voraus». Ist dem so?** Nein. Die neueste Version des iPhones, das 3G S, korrigiert nur Versäumnisse des alten Gerätes: bessere Kamera, Kompassfunktion, schnellerer Prozessor. **Ein unglücklicher Slogan also.** Der Fokus liegt auf der Software. Nach nur einem Jahr sind über 50000 Applikationen bei iTunes erhältlich. Monatlich kommen 1000 hinzu. Neu ist auch die Schnittstelle für Peripherie-Geräte am iPhone offen. So kann ich Gadgets direkt mit dem Telefon verbinden. **Zum Beispiel?** Ein Entwickler hat das Handy zwischen elektrischer Gitarre und Verstärker geschalten und als Effektgerät benutzt. Ein anderer hat es an den Bordcomputer seines Autos gehängt, um den Benzinverbrauch zu kontrollieren. **Wie haben Sie am meisten profitiert? Die Teilnahme kostet immerhin 1295 Dollar.** Das würde ich gerne erzählen. Aber Apple hat eine mehrmonatige Sperrfrist verhängt. Das ist ärgerlich. Wir Programmierer können nicht öffentlich Erfahrungen austauschen. Apple öffnet sich nur halb.

[1] http://www.sonntagszeitung.ch/

[2] /press/SonntagsZeitung-WWDC.pdf

«Apple öffnet sich nur halb»

Schweizer Entwickler berichtet von WWDC

Adrian Kosmaczewski an der Apple-Konferenz, San Francisco

Das Motto der Worldwide Developer Conference von Apple lautet «ein Jahr später, Lichtjahre voraus». Ist dem so?
Nein. Die neueste Version des iPhones, das 3G S, korrigiert nur Versäumnisse des alten Gerätes: bessere Kamera, Kompassfunktion, schnellerer Prozessor.

Ein unglücklicher Slogan also.
Der Fokus liegt auf der Software. Nach nur einem Jahr sind über 50 000 Applikationen bei iTunes erhältlich. Monatlich kommen 1000 hinzu. Neu ist auch die Schnittstelle für Peripherie-Geräte am iPhone offen. So kann ich Gadgets direkt mit dem Telefon verbinden.

Zum Beispiel?
Ein Entwickler hat das Handy zwischen elektrischer Gitarre und Verstärker geschaltet und als Effektgerät benutzt. Ein anderer hat es an den Bordcomputer seines Autos gehängt, um den Benzinverbrauch zu kontrollieren.

Wie haben Sie am meisten profitiert? Die Teilnahme kostet immerhin 1295 Dollar.
Das würde ich gerne erzählen. Aber Apple hat eine mehrmonatige Sperrfrist verhängt. Das ist ärgerlich. Wir Programmierer können nicht öffentlich Erfahrungen austauschen. Apple öffnet sich nur halb. BARNABY SKINNER

Best WWDC Ever

Adrian Kosmaczewski

2009-06-13

… and WWDC 2009 is finally over.

This year's event has been nothing short of amazing; maybe because not only the technologies presented blew my mind, but also because I met and spent some time with incredible guys, and getting in touch with the right people changes everything. So, to all of you, many thanks: @cigumo[1], @dlpasco[2], @sophiestication[3], @davemark[4], @jeff_lamarche[5], @markuspalmanto[6], @serpah[7], @raminf[8], @geraudch[9], @ayasin[10], @octopus_prime[11], @pjay_[12], @2009wwdc[13] and all the others, in and out of Twitter, like Julio from Guatemala, the guy from Adobe (met in the queue to the hotdogs in the beer bash of Yerba Buena gardens), Sandro (aka "The Crazy Swiss Guy" of the Stump the Experts session), etc, etc, etc… with whom we've shared laughs, ideas, emotion, friendship and beers.

WWDC is an inspiring event: listening to the above guys, or the conferences from Smule[14] or ngmoco:)[15] talking about their companies, and how they grew up the past year, all of that makes me think about this new path I'm taking right now:

akosma software[16] **is born.** Expect a lot.

[1] http://twitter.com/cigumo
[2] http://twitter.com/dlpasco
[3] http://twitter.com/sophiestication
[4] http://twitter.com/davemark
[5] http://twitter.com/jeff_lamarche
[6] http://twitter.com/markuspalmanto
[7] http://twitter.com/serpah
[8] http://twitter.com/raminf
[9] http://twitter.com/geraudch
[10] http://twitter.com/ayasin
[11] http://twitter.com/octopus_prime
[12] http://twitter.com/pjay_
[13] http://twitter.com/2009wwdc
[14] http://www.smule.com/
[15] http://blog.ngmoco.com/
[16] http://akosma.com/

Video Interview on TUAW

Adrian Kosmaczewski

2009-06-15

Interview by Brett Terpstra published by The Unofficial Apple Weblog[1].

[1] http://www.tuaw.com/2009/06/16/wwdc-live-adrian-kosmaczewski/

Myself on the Swiss Press

Adrian Kosmaczewski

2009-06-16

This is something that does not happen that often to me, so it deserves a blog post of its own: here's two appearances of yours truly in the Swiss press last week. The first one is an article in the Tages Anzeiger of last Wednesday (full text available online[1]), and the second one is an article (with photo) on the SonntagsZeitung[2] last Sunday. Since the SonntagsZeitung article is not available online, here's the scan of the article[3].

Yay! :)

[1] https://akos.ma/press/TagesAnzeiger-WWDC.pdf
[2] http://www.sonntagszeitung.ch/
[3] https://akos.ma/press/SonntagsZeitung-WWDC.pdf

Myself on TUAW

Adrian Kosmaczewski

2009-06-17

No comments :)

Thanks to Brett Terpstra (who interviewed me) and the TUAW guys for publishing the interview!

OpenGL ES 2.0 on iPhone OS 3.0

Adrian Kosmaczewski

2009-06-24

Now that the NDA on the iPhone OS 3.0 SDK has been lifted[1] (which happened much faster than what I thought it would take!) here's my first contribution to the world of iPhone OS 3.0 open source code: sample code about how to use OpenGL ES 2.0 on the iPhone 3GS, something I announced in Twitter[2] last week.

As you might know by now, one of the biggest enhancements (and yet, one of the most obscure[3]) of the newly released iPhone 3GS[4] is the new GPU chipset, which allows developers to create applications using Open GL ES 2.0 (together with Open GL 1.1, which was already available in the first two iterations of the iPhone). This is a major advance, invisible to the end user, which, coupled with the unprecedented performance boost[5] of the iPhone 3GS, opens up the possibility to developers to create applications with new textures and effects, yet unforeseen on this platform.

Given that Xcode does not (yet) bring an Xcode template to play with, and that the OpenGL ES 2.0 Programming Guide book, by Aaftab Munshi, Dan Ginsburg and Dave Shreiner[6] does not (obviously) bring iPhone examples, I have created a project in Github[7] where I will be publishing the code samples in the book, as I progress in the lecture[8], ordered by chapter, ready to compile and play with.

Enjoy! I'm happy not having to use the word "[REDACTED]"[9] any more now (there's the other OS, the bigger cat, but, oh well…)

Update, 2009-06-24: I just found this blog post[10] by the folks of

[1] https://devforums.apple.com/message/86763#86763
[2] http://twitter.com/akosma/statuses/2181419577
[3] http://twitter.com/akosma/status/2082319555
[4] http://www.apple.com/iphone/
[5] http://www.theiphoneblog.com/2009/06/23/raw-performance-iphone-3gs-4x-faster-iphone-3g-3d/
[6] http://www.amazon.com/OpenGL-ES-2-0-Programming-Guide/dp/0321502795
[7] http://github.com/akosma/opengles2-xcode-book/
[8] http://twitter.com/akosma/status/2092228652
[9] http://twitter.com/akosma/status/2107210213
[10] http://blackpixel.com/blog/244/early-iphone-3g-s-opengl-test-results/

Black Pixel[11] (Daniel Pasco[12]'s company) with benchmarks about OpenGL ES on the iPhone 3GS... and the first line says it all:

> Holy crap, this thing is fast

Update, 2009-06-24: Jeff LaMarche just announced[13] that the authors of the book have published iPhone - compatible[14] code in the book website[15]! That effectively renders this project useless :))

[11] http://blackpixel.com/

[12] http://twitter.com/dlpasco

[13] http://iphonedevelopment.blogspot.com/2009/06/opengl-es-20-programming-guide-has-3gs.html

[14] http://opengles-book.com/downloads.html#iPhone

[15] http://opengles-book.com/

WordPress 2.8 and the get_link() error in line 647 of dashboard.php

Adrian Kosmaczewski

2009-06-25

Wow, that's a long title, but it should drive people with this problem right here. If you have upgraded your WordPress installation to 2.8, you might have encountered a nasty error in your Dashboard, which says something about a

> Fatal error: Call to a member function on a non-object in /home/user/www/wp-admin/includes/dashboard.php on line 647

This has been reported in the WordPress site[1] but no fix has been provided. I found elsewhere a possible fix[2], but in my case, the new URL would not be saved at all, and the problem would persist.

I fixed it using a good old method, enabled by Open Source©: **editing the code directly**; I'm posting it here for those of you who would like to do it, until 2.8.1 is released:

```
$author = $item->get_author();
if ($author == NULL)
{
    $site_link = "";
    $publisher = "Someone";
}
else
{
    $site_link = esc_url(strip_tags($author->get_link()));
    if ( !$publisher = esc_html(strip_tags($author->get_name())))
        $publisher = __('Somebody');
    if ($site_link)
        $publisher = "<a href='$site_link'>$publisher</a>";
    else
        $publisher = "<strong>$publisher</strong>";
}
```

[1] http://wordpress.org/support/topic/279727

[2] http://estanli.net/blog/2009/06/24/wordpress-2-8-problem-fatal-error-call-to-a-member-function-on-a-non-object/

Interview sur Le Temps

Adrian Kosmaczewski

2009-07-14

Interview by Anouch Seydtaghia published in Le Temps[1] (download PDF of the website article[2] or the PDF of the printed article[3]).

> Tout comme Stephan Burlot, Adrian Kosmaczewski, développeur à Genève, n'est pas prêt à développer des applications pour d'autres plateformes: «Je le ferai sans doute lorsque les téléphones Android de Google seront davantage présents sur le marché, mais pas avant. Je ne veux pas me disperser et préfère me concentrer sur l'iPhone, clair leader du marché.»

[1] http://letemps.ch/Page/Uuid/b528c25e-6fed-11de-813a-c70b39535a9f/Apple_reste_intouchable_sur_le_march%C3%A9_des_applications

[2] http://akosma.com/press/LeTemps_web_Juillet_2009.pdf

[3] /press/LeTemps_print_Juillet_2009.PDF

bluewoki - Bluetooth Walkie-Talkie for iPhone OS 3.0

Adrian Kosmaczewski

2009-07-15

The first iPhone application under the akosma[1] brand has just been published on the App Store:

Available on the App Store[2]

bluewoki[3] is a very simple walkie-talkie application, which uses the new GameKit framework available on the iPhone OS 3.0. It is available at the App Store[4]! In English, Spanish and French.

The source code is really simple (around 170 lines of code) and you

[1] http://akosma.com
[2] http://itunes.com/apps/bluewoki
[3] http://bluewoki.com/
[4] http://itunes.com/apps/bluewoki

can get it, as usual, from Github[5] with a BSD license.

[5] http://github.com/akosma/bluewoki/

Objective-C Compiler Warnings

Adrian Kosmaczewski

2009-07-16

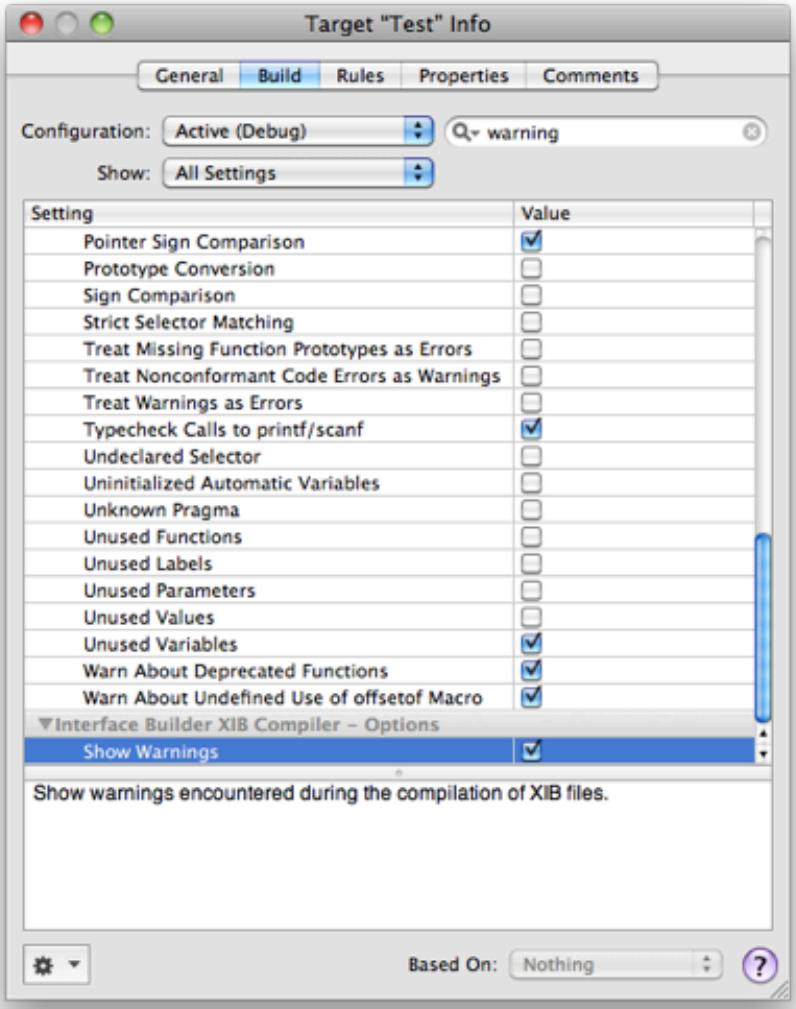
A recent comment[1] by Joe D'Andrea[2] in a previous post[3] reminded me about the importance of removing compiler warnings in Xcode projects. Most importantly, it reminded me of a conversation with a fellow developer a couple of weeks ago, in which he told me that he was surprised to see that my projects compiled all the time without warnings. Not a single one. Nada. And that I took the time to remove them before checking code into source control.

He actually didn't know you could remove all compiler warnings; he thought Objective-C was the land of compiler warnings. This situation, I think, is far from exceptional, and due mostly to cultural and technical reasons.

It is my opinion, that removing compiler warnings is **basic project hygiene**, like writing unit tests, or using the Clang Static Analyzer. I will explain in this post some techniques I use to remove warnings in my Objective-C code.

---

[1] /blog/opengl-es-2-on-iphone-os-3/#comment-26078
[2] http://twitter.com/jdandrea
[3] /blog/opengl-es-2-on-iphone-os-3/



First of all, why does the Objective-C compiler (or compilers in general) output “warnings”? Many developers are puzzled the first time they encounter them, since even if the compiler complained, the application usually runs anyway without (perceptible) problems.

Warnings are used to signal specific issues in the source code which could potentially lead to crashes or misbehavior under some circumstances, but which should not (pay attention to the verb “should”) block the normal compilation and (hopefully) execution of your code (otherwise, it would be a compiler error).

It’s the way used by your compiler to say:

> Hey, I’m not sure, but there’s something fishy in here.

Not removing warnings, as I said above, is a problem that originates both in the programming background of the developer, and specific technical issues.

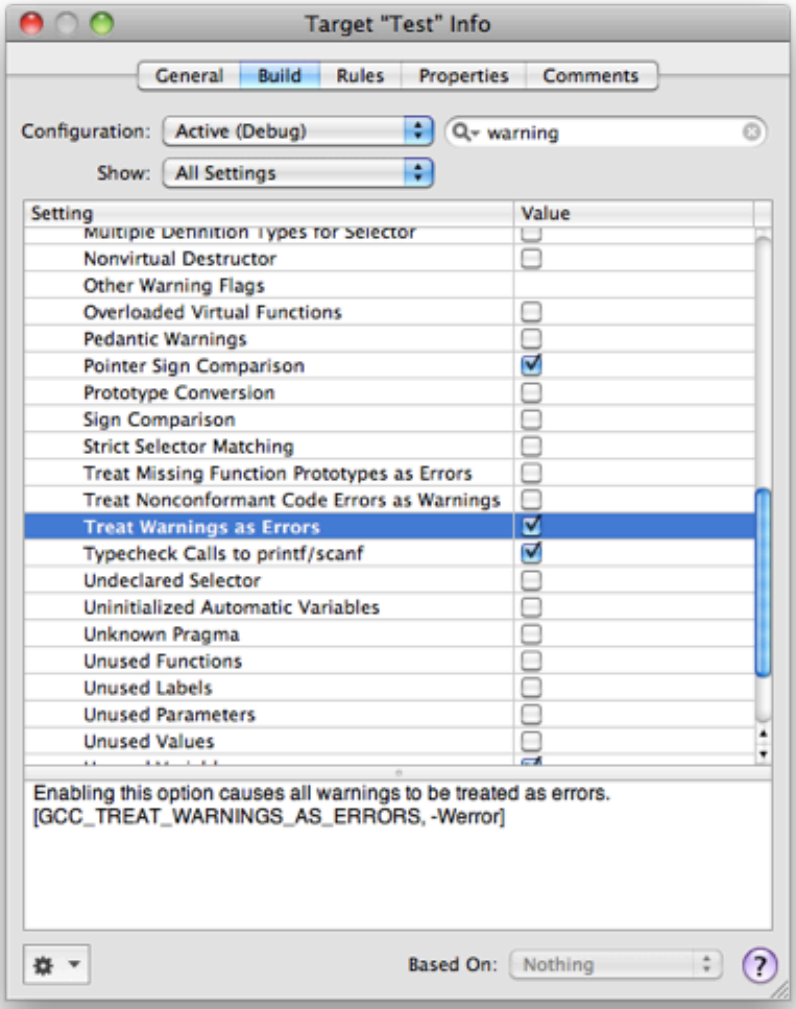
Culturally speaking, many other programming environments either do not have compilers at all (at least not “visible” ones, like Ruby or PHP) or simply do not spit warnings for anything else than deprecated methods (like C# or Java); this situation has made many developers new to the iPhone platform to blatantly ignore them.

Technically, given the fact that Objective-C is the “other” object-oriented superset of C, and that it behaves as a coin with both a static and a dynamic side, compiler warnings convey a great amount of precious information that must *never* be ignored.

In this sense, Objective-C has a lot in common with C++. Ignoring warnings in C++ is strongly discouraged, and Scott Meyers explains this in chapter 9 of his book “Effective C++”, stating that (third edition, page 263):

> Take compiler warnings seriously, and strive to compile warning-free at the maximum warning level supported by your compilers

In the case of Objective-C, this can be done by setting `GCC_TREAT_WARNINGS_AS_ERRORS` (-Werror) to true in your build settings.



Steve McConnell takes this advice to another level of importance in his classic book “Code Complete” (second edition, page 557):

> Set your compiler’s warning level to the highest, pickiest level possible, and fix the errors it reports. It’s sloppy to ignore compiler errors. It’s even sloppier to turn off the warnings so that you can’t even see them. Children sometimes think that if they close their eyes and can’t see you, they’ve made you go away (…).
>
> Assume that the people who wrote the compiler know a great deal more about your language than you do. If they’re warning you about something, it usually means you have an opportunity to learn something new about your language.

To give a concrete example of the importance of warnings, many of us have had to migrate applications developed for iPhone OS 2.x to the 3.0 operating system, mostly because failure to run on the new version of the OS was ground for removal from the App Store. That moment of truth, the rebuild of the Xcode project, unveiled a plethora of compiler warnings, most due to deprecated methods, like the tableView:accessoryTypeForRowWithIndexPath: method of the UITableViewDelegate protocol, or the initWithFrame:reuseIdentifier: method of the UITableViewCell class (which, incidentally, are properly marked as such in the documentation, too).

Compiler warnings in Objective-C have a multitude of reasons:

- Using deprecated symbols;
- Calling method names not declared in included headers;
- Calling methods belonging to implicit protocols;
- Using some ambiguous commands which might be intentional but are syntactically valid anyway;
- Forgetting to return a result in methods not returning “void”;
- Forgetting to #import the header file of a class declared as a forward “@class”;
- Downcasting values and pointers implicitly.

Many solutions exist for these problems, and I do not claim to know them all; I’ll just describe some of them, and hopefully some of the readers of this post will add others in the comments below.

1) Make implicit protocols explicit

This is a really simple one, and it’s also good for documentation and code clarity reasons. Get all the references of delegate methods you use in the code, and group the methods into their own header file. Import the header file whenever you need, and make classes explicitly implement them:

```
//...
@interface NewClass : NSObject <SomeProtocol>
{
    //...
```

Of course, take advantage of Objective-C’s @required and @optional keywords in your protocol declarations; they are used by the compiler to verify (or not) the existence of delegate methods in your class implementation. This way, you’ll surely remove some warnings.

2) ~~Do not use “id as the data type for delegate fields~~ Using id as datatype for delegate fields

I personally use the following declaration for delegate fields:

```
//...
NSObject<SomeProtocol> *delegate;
```

```
//...
```

instead of simply

```
//...
id delegate<SomeProtocol>;
//...
```

This is because I always check on delegates before calling their methods. Call me paranoid, but this is what a good delegate call looks to me:

```
if([delegate respondsToSelector:@selector(someObj:doesThis:)])
{
    [delegate someObj:self doesThis:@"123"];
}
```

Using NSObject instead of id in the delegate declaration avoid yet another warning. **Update, 2009-07-17:** This is because using `id<SomeProtocol>` raises a warning that "respondsToSelector:" is not defined in SomeProtocol. The solution for this is making SomeProtocol inherit from the NSObject protocol (I always forget this double life of the NSObject symbol):

```
@protocol SomeProtocol <NSObject>
//...
@end
```

This way, you can use id variables without problem.

3) Create categories for private methods

Objective-C uses the @private, @public and @protected identifiers only for instance fields, but otherwise methods can only be marked as instance ("-") or static ("+"), but all methods specified in the header file are public by default.

If you need to specify private methods, do that in your implementation file, creating what's called a "category" of your own class, which basically "extends" the class with new methods:

```
#import "NewClass.h"

@implementation NewClass (Private)
// your methods here
@end

@implementation NewClass
// ...
@end
```

This way you can define private methods, not exposing them in the public interface file. And you remove some more warnings.

4) Turn implicit type conversions and casts into explicit ones:

This one is inspired by McConnell's Code Complete (second edition, page 293):

```
int i;
float y, x;
y = x + (float)i
```

Even if the compiler could work out the "y = x + i" expression without problem, the code above will remove yet another warning, and will make your code more obvious and easier to read, since it clearly states your intentions.

5.1) Support earlier OS versions via runtime checks

If you have to write iPhone applications compatible with both the 3.0 and 2.x versions, this sample from Apple https://developer.apple.com/iphone/library/samplecod provides instructions on how to do it.

"MailComposer runs on earlier and later releases of the iPhone OS and uses new APIs introduced in iPhone SDK 3.0. See below for steps that describe how to target earlier 0S versions while building with newly released APIs."

It is worth noting that this sample application compiles without a single warning!

5.2) Support earlier OS versions via #defines

Use #ifdef IPHONE_OS_3.0 and IPHONE_OS_2.2.1 if you can (or must) provide different binaries for each supported platform. This might be the case for in-house applications, but again, it might help removing some warnings too.

6) Use @class in the @interface, #import on the @implementation

Whenever you use a class on a header file, to avoid cross-references, use the @class keyword to reference it, but do not forget to #import its header file in the implementation!

```
@class AnotherClass

@interface SomeClass : NSObject
{
@private
    AnotherClass *field;
    //...
}
```

and then

```
#import "AnotherClass.h"

@implementation SomeClass

- (id)init
{
    if (self = [super init])
    {
        field = [[AnotherClass alloc] init];
    }
    return self;
}

@end
```

The problem is, if you do not #import the file, you get a warning…

The advantage of this technique is not obvious in small projects, but it is strongly recommended anyway; it prevents header files from cross-referencing each other, and it reduces build times in projects anyway. It is the Objective-C analog to the technique of using "class" statements in C++ header files, instead of #include.

For more information: Matt Gallagher also wrote about this issue, and I strongly recommend his article[4] too! Read also this article from Apple[5] about the issue of compiler warnings which covers all the options available on GCC in great detail.

Finally, all of this boils down to the fact that iPhone programming is not as easy as web development[6], and that iPhone applications, for many reasons, require patience and attention. Removing warnings from your code is just one of many steps[7] to have great iPhone applications, running smoothly without problems.

Update, 2009-07-17: Regarding the "id vs. NSObject" issue, I've used NSObject because you get a warning when calling respondsToSelector: on a variable of type id. Now, after reading all the comments I've gone back to Xcode and found out that you can define SomeProtocol as implementing the NSObject protocol itself, and this solves the problem. Thanks everyone for the heads-up!

[4] http://cocoawithlove.com/2009/04/8-confusing-objective-c-warnings-and.html
[5] http://developer.apple.com/Tools/xcode/compilercodewarnings.html
[6] /blog/dirty-little-secret/
[7] /blog/10-iphone-memory-management-tips/

Blessed

Adrian Kosmaczewski

2009-07-17

I'm blessed.

This post is about life, about the tiny little things that make our current world a great, truly enjoyable experience. Grab a cup of tea, sit and relax on your chair, and read. This is clearly a blog post suitable for a Friday afternoon.

I remember the day ENTEL (Argentina's former national telecom company, back in the 80's) installed a telephone line at home. I remember that day very well, because I was 16 years old. That's right, I lived without a phone for most of my childhood(*). No phone rings, no answering machines, no dialing pads, no twisted black rubber cables, nothing. Cellphones were just science fiction. Really. Just like video conferences and handheld communicators.

We at Argentina we were so used to not having a phone, that I remember all the little tricks we had to use with friends to meet up, or with my mother to keep her updated on my whereabouts. For example, it was common for people to just drop by, ring and see if you were there for a mate on a Sunday afternoon. There was no way they could tell you they wanted to meet you (other than by regular mail or telegram, which is kinda weird anyway), so dropping by was the only option left.

And nobody complained about it. It was cool, it was pure surprise.

OK, OK, not having a phone was not always cool, particularly when my grandmother fell ill in 1985 and calling for an ambulance became an almost impossible feat; my mother had to run to the nearest bar to call for it, since there weren't public payphones nearby - at least not working ones.

But we managed to live a happy life. It might sound ridiculous, but I never thought of the fact of not having a phone like a major issue. An annoyance at most, but not a major problem.

And the day we received our phone, it looked like this:

(source)

I think the picture speaks for itself. That was at the end of 1989, a couple of weeks before the Berlin wall collapsed, and right in the middle of one of the worst hyperinflations known to modern economists.

I also remember the day the phone started to be useful, too. That was like 3 months after they installed the line, but that's another story. Suffice to say that, one day, the phone rang. Neither my mother nor me knew what to say when picking it up.

(really) Fast forward 20 years later, I'm living in Switzerland, I'm married to the lovely love of my life, I live in an appartment with a big window over Lake Geneva, and I write software applications for a small, incredible phone.

A lot has happened in the middle. A lot.

I remember my mother struggling to bring food home back in the 80's. As simple as that. She worked as an employee in a state-run company, which was privatized a couple of years after we left the country. Living in Argentina in the 80's was a difficult task, with incredible inflation and low salaries, and it mostly consisted in buying dollars (in the black market) the day you got your payslip, and selling them (in the black market too) whenever you needed cash.

My mother also used to play with credit cards, which at the time were not tied to inflation rates; that meant that you could buy today in the

supermarket for 200 pesos of food, which was around (say) 10 dollars, and pay the following month, when the same 200 pesos (+ interests) would be changed for 2 dollars.

These tricks were not always enough. And some times, in spite of all the efforts of my mother, the fridge would be particularly empty, to the greater sorrow of this brave woman. Now that I'm an adult, I begin to understand the level of desperation she had to suffer, and the enormous effort she made every evening to have a smile on her face.

One day, back in February 1991, my mother and I left Argentina for Switzerland. She had been in this country twice, in 1954 and 1963. I had never been here; heck, I had never even left the country before. Not even to Uruguay.

I remember hearing people speak in French and English for the first time in my life. I remember not getting a word of what people said on TV or the radio. I remember not getting all the jokes my schoolmates said during breaks, and I remember getting myself into trouble for mistranslating words or phrases.

I have a Polish family name, and two passports; my father lives in Buenos Aires and my mother lives in Geneva. My wife comes from Bolivia. I am grandson of migrants who travelled accross the Atlantic back in the 20's and the 30's. I am son of a migrant who did the journey back 60 years later.

I am a migrant. My wife is a migrant. My family is a family of migrants.

Even if I can freely move through the customs offices of most major airports, I feel as a migrant, and I understand the pain that comes with having to leave part of your life behind, of being different, of dealing with incomprehension and being an outsider.

I can't support "northern" macro immigration policies, just because I lived the process at micro level. I can't justify censorship, because I know the cost of communication. I can't support dictatorship, because I've lived in one, and I remember it very well.

I remember when my mother went to the PTT (before Swisscom existed) to ask for a phone line for our first appartment in Geneva. The woman at the counter asked her when did she prefer the technician to come by our place.

My mother clearly did not expect that question.

The guy came in the following day, and when he left, my mother picked up the phone, and incredibly enough, a tone sounded. The look in her eyes said it all.

However, even if we had a phone, and even if my mother called some cousins to tell them she was back in Switzerland, just one of them called back. Even worse, the cost of international communications at the beginning of the 90's was prohibitive, and we slowly lost contact with many friends in Argentina.

We had a phone, but not many people to talk to. That was the major tradeoff.

Somehow, many pieces of my life (people, meetings, encounters, places) have come accross in a delicious yet uncanny way, assembling a whole that amazes me, in this very day.

And that's why I say I'm blessed. As we all are. We have food in our plates, we have phones ringing with friends on the other side of the line, heck, we have e-mail, Facebook, Twitter, GPS, iPhone, Google, Skype - which effectively means that yes, we live in a sci-fi world with video conferencing and handheld communicators. Communication is a commodity. Information flows a gazillion times faster than it did 20 years before.

The infinite possibilities you are offered in this world are our greatest asset. Many times we forget this, but here I am, humbly reminding me (and my readers) about this very fact:

We are blessed. Now pick up your phone, open up your Skype or fire up your e-mail client, and call or write a friend with whom you haven't spoken for a while.

(*) For those wondering, my mother had moved to that appartment in 1970, and asked for a phone at that time. In 1985 she subscribed to the "plan Megatel", created by former argie president Raúl Alfonsín to create 1 million phone lines before the end of his term in 1989. At about 25 dollars per month, that phone costed her about USD 1500 until 1990. Check this article (in Spanish) for more information.

Code Organization in Xcode Projects

Adrian Kosmaczewski

2009-07-28

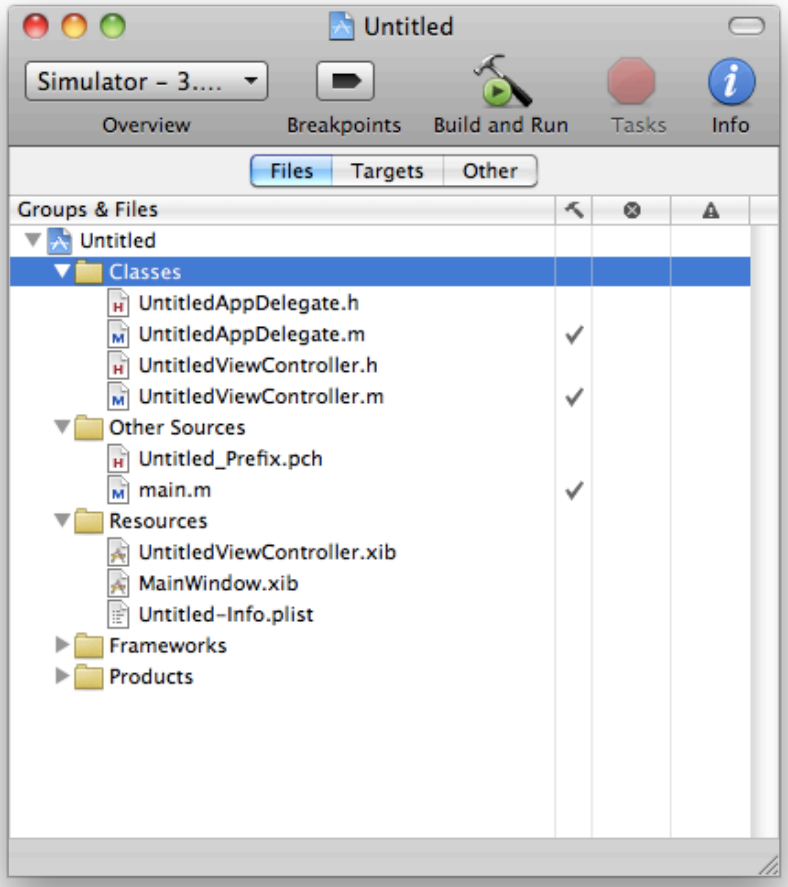
Xcode does not impose any structure to your source code tree. This is both cool and useful to quickly throw a couple of lines for a prototype, but in my experience, this approach does not scale. More often than not, without any hygiene, your project can become a mess. Just using Xcode defaults, after a while your resources will sit beside your .xcodeproj file, all the project classes will be thrown together in the Classes folder, and if you have a relatively large project, this approach makes finding individual files painful.

Of course, Xcode provides “Groups” to organize your source code, but the idea is to be able to quickly identify the different kind of files that make up your Xcode project, either for Mac or for the iPhone, without having to open the Xcode project file. This means having both a folder structure, and an internal source code file structure. All of this will help you maintain your project in the future, which means cheaper costs, and less time spent looking for bugs.

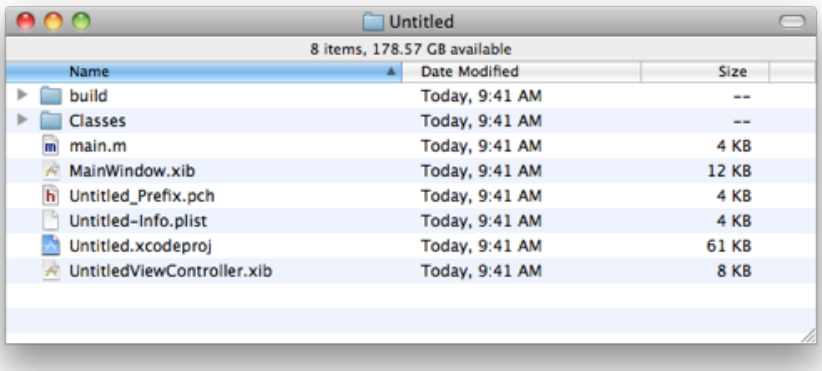
All of this is also particularly useful when browsing projects via Google Code, Github or any other kind of file view of source code repositories. If your code is organized in a nice folder structure, it is easier to explore than if all the files sit in the same folder.

In this post I will enumerate some best practices that I use in all of my projects.

So let’s say that you start a new Xcode project. Here’s the Xcode window that is presented to you (seen in “Condensed” mode, which is the one I prefer):



This is the project layout as seen in the Finder:



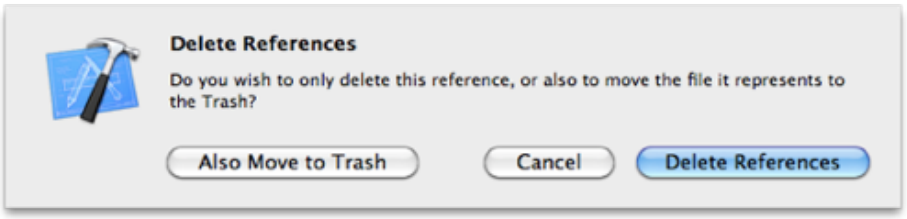
As you can see (and as you might have experienced), in the default

layout used by Xcode, all new source code files will be thrown into the “Classes” folder, while all the new Resources will be just stored in the project root. In the long term, this layout can be really painful to deal with. So let’s just start rearranging things a bit:

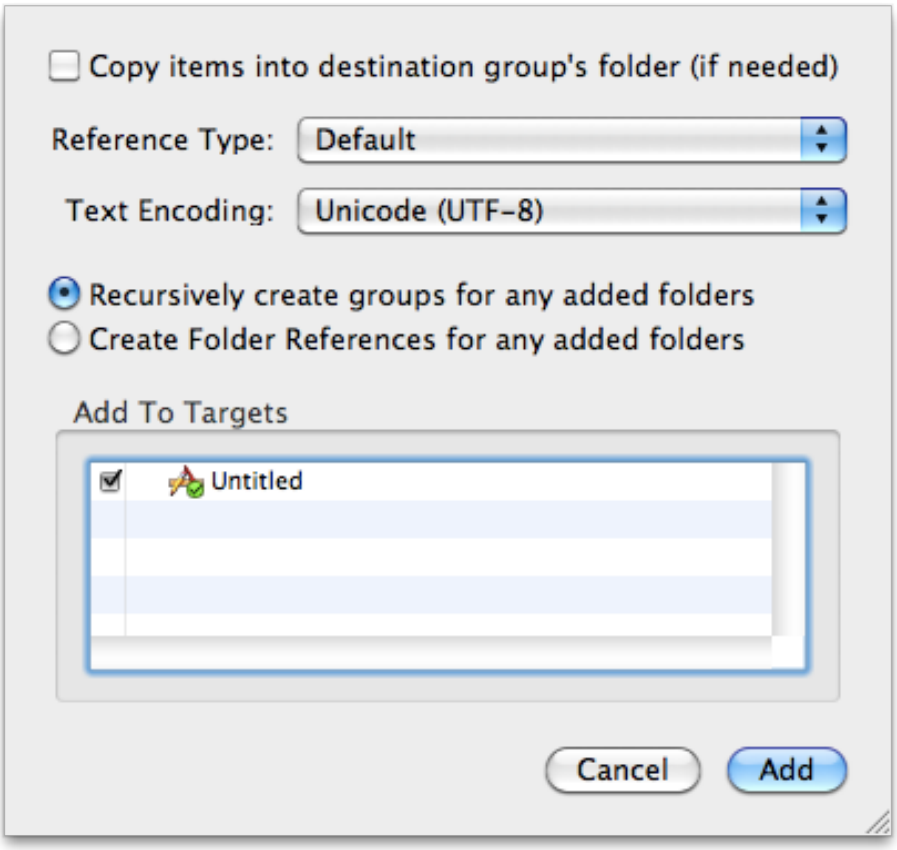
Organize source files in folders and mirror them in Xcode

When I start a new Xcode project, I usually do the following:

1. I remove the “Classes” group (just deleting references, not moving the items to the trash, as shown in the image below)



2. Then I add the following subfolders to the “Classes” folder:
    - AppDelegate
    - Controllers
    - Models
    - Helpers
3. Finally, drag the enhanced “Classes” folder from the Finder to the Xcode project window, asking to “recursively create groups for every subfolder”:



Create separate folders for different Resource elements

The next step is to actually create a resource folder in Finder, and add a subfolder for every kind of resource that my project will use: sounds, images, SQLite databases, NIBs, etc. Then I do the following:

1. Remove the Resource group from the Xcode project (just deleting references)
2. Then I drag the newly created “Resources” folder from the Finder to the Xcode project window, asking to “recursively create groups for every subfolder”, like we did for the “Classes” folder.

Doing this has an interesting side effect: when you localize your application in other languages, each folder will contain a subfolder with the localized resources inside (for example, an “en.lproj” for English, “es.lproj” for Spanish, and so on).

Organize your code consistently

Each @implementation *.m file should always present methods in this order:

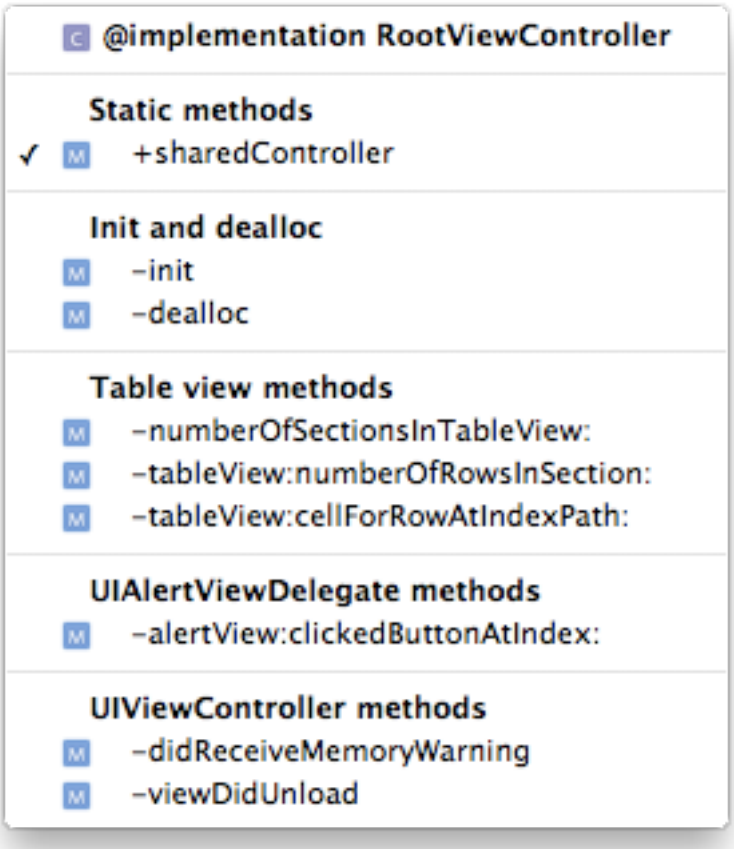
1. init and dealloc
2. public methods
3. public @dynamic properties
4. delegate methods (for each supported protocol)
5. private methods

Use #pragma statements to separate the regions shown above

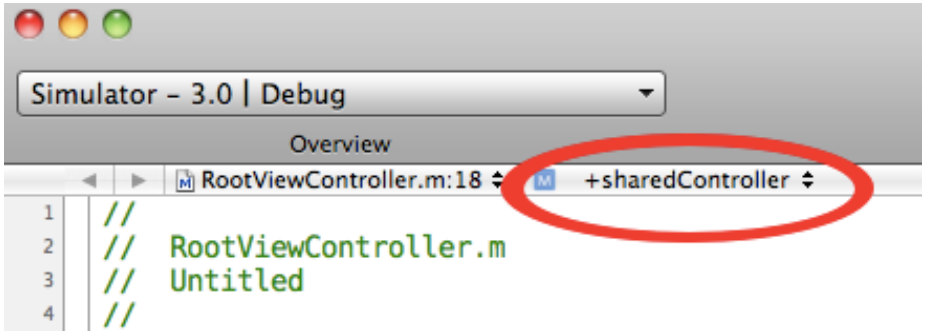
Each logic group of methods should be separated from each other using the following lines (just type "#p" and hit the TAB key in Xcode!)

```
// ...
    return cell;
}

#pragma mark -
#pragma mark UIAlertViewDelegate methods

- (void)alertView:(UIAlertView *)alertView
        clickedButtonAtIndex:(NSInteger)buttonIndex
{
// ...
```

The advantage of this approach is that later, you can use those #pragma marks to generate an automatic layout in the symbols pop-up of Xcode:



You can get this pop-up window clicking on this sector of your Xcode window:



Only leave public methods in the header files

This means putting private methods definitions in a (Private) category on top of the *.m file. This will remove all compiler warnings (about "this class might not respond to this selector") and will cleanly separate what's public from what's not:

```
#import "UntitledViewController.h"

@interface UntitledViewController (Private)
- (id)returnPrivateObject;
- (void)changeInternalState:(NSString *)param;
@end

@implementation UntitledViewController

- (id)init
{
```

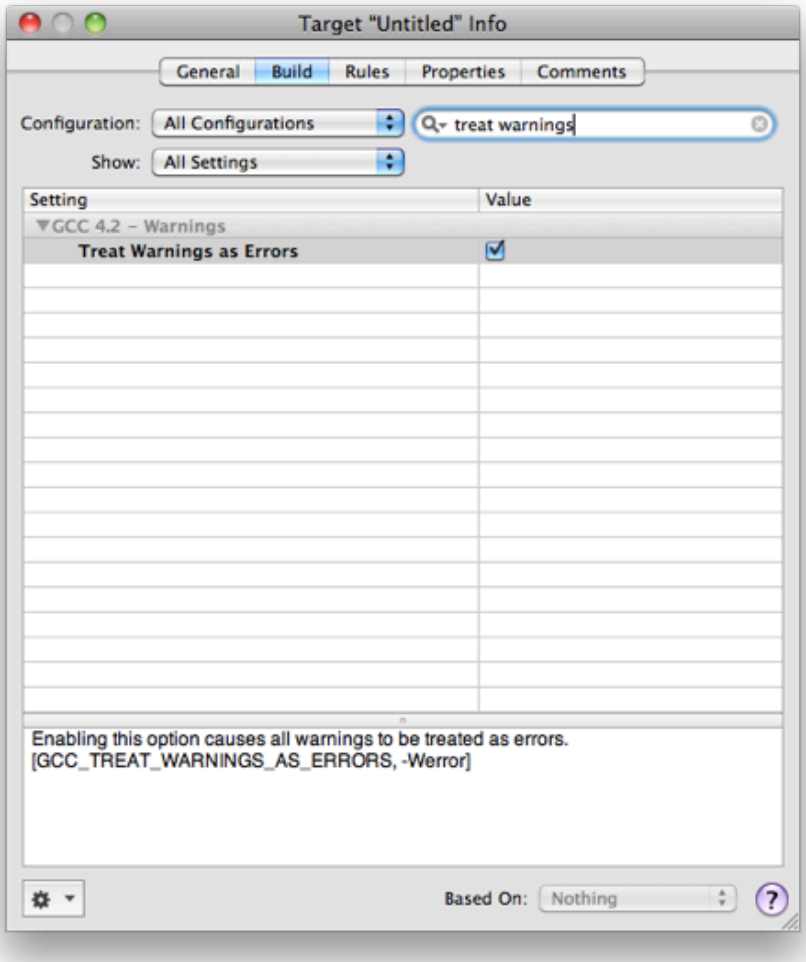
Use consistent coding conventions

You can use my own[1], if you wish.

Treat warnings as errors

I've said a lot about that in a previous post[2], but here's a quick reminder:

[1] http://wiki.akosma.com/Objective-C_Code_Standards
[2] /blog/objective-c-compiler-warnings/



Create “Distribution” configurations for the new project

Duplicate the “Release” configuration and create two new ones: “Distribution Ad Hoc” and “Distribution App Store”. Each one will have to be configured with their corresponding provisioning profiles.

Add an “Entiitlements.plist” file to the project

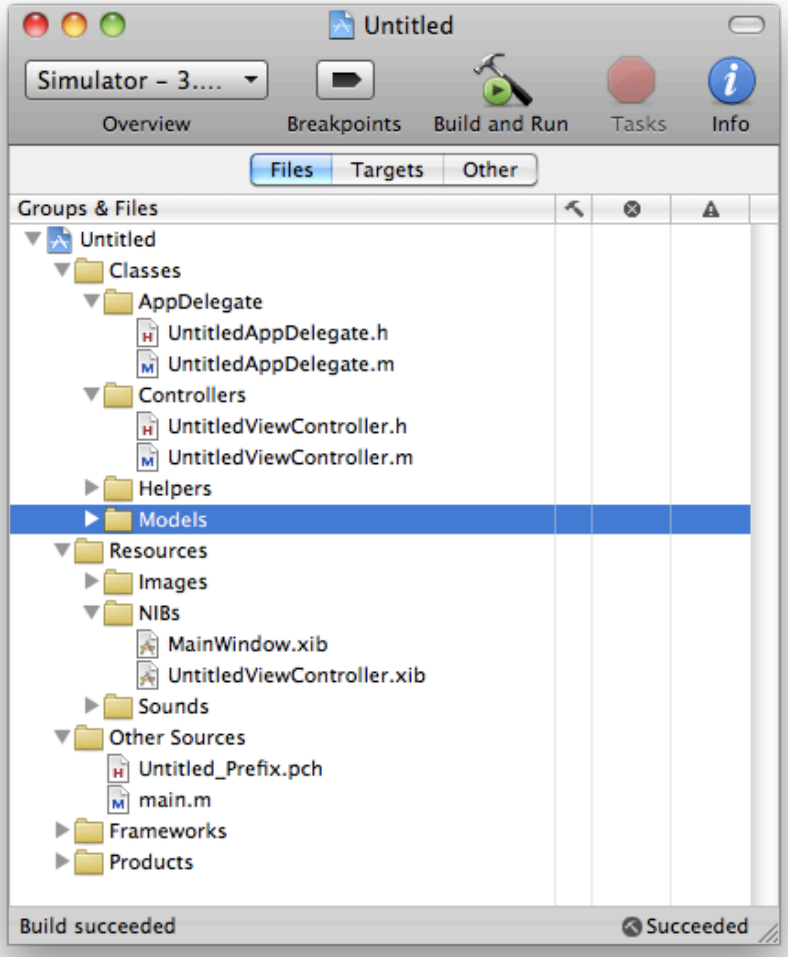
Remember to uncheck (disable, turn off) the “get-task-allow” value (I still don’t understand why every Xcode project does not create this file automatically). Then add the “Entitlements.plist” value to the corresponding key in the “distribution” configurations created in the previous step.

Use source control

As soon as your project source tree is ready, commit it to your repository, whichever this is.

Conclusion

This is how the final project might look like:



Of course, you might as well enforce the above best practices using your own default project templates; depending on your requirements, this might be a useful thing to do. You must store those new Xcode templates in the following locations:

- **iPhone:** /Developer/Platforms/iPhoneOS.platform/Developer/Library/Xcode/Project Templates

- **Mac:** /Developer/Library/Xcode/Project Templates

Hope this helps! As usual, feel free to add your comments, best practices, rants and other reactions in the comments section below.

Risk Management in iPhone Projects

Adrian Kosmaczewski

2009-08-03

Let's be frank: **it's not the best time to be an iPhone developer right now**. In just one year of existence, the App Store seems to have evolved from the hottest[1] to the lamest[2] status, without any time to breathe in the middle, but with some warning[3] signs[4] every so often.

Several iPhone developers have publicly stated their opposition to the Google Voice fiasco[5] (starting with Riverturn themselves, the developers of the application), and many have simply stopped creating iPhone OS applications altogether; just to name a few, Fraser Speirs[6], Steven Frank[7] and Andrew Wulf[8] have publicly stated that they don't want to deal with the App Store process anymore. And I'm sure that there are many more developers evaluating this very possibility out there; when you have Om Malik[9] or Michael Arrington[10] bashing the iPhone, it sure creates a lot of buzz and uncertainty in the market.

However, and this is my official position, **even if I do not agree with the current App Store policies, I'm not quitting the iPhone OS**

[1] http://www.crunchgear.com/2009/04/04/lets-all-quit-our-jobs-and-become-iphone-app-developers/

[2] http://www.riverturn.com/blog/?p=455

[3] http://almerica.blogspot.com/2008/09/podcaster-rejeceted-because-it.html

[4] http://www.tuaw.com/2009/06/12/app-store-lessons-the-game-changer-rejection/

[5] http://www.riverturn.com/blog/?p=455

[6] http://speirs.org/2008/09/12/app-store-im-out/

[7] http://stevenf.tumblr.com/post/152606616/important-note-references-to-i-in-this-post

[8] http://thecodist.com/article/kissing_the_app_store_goodbye

[9] http://gigaom.com/2009/02/11/my-big-iphone-break-up/

[10] http://www.techcrunch.com/2009/07/31/i-quit-the-iphone/

platform anytime soon. I'll build more applications for the iPhone in the future – heck, I've got 2 already[11] approved[12] and 3 more on the approval process pipeline, with at least 3 more in the development phase. **My plan, and what this article is about, is about managing the risk represented by Apple in this business.** It might be hard, but it's not impossible, no matter what others say[13].

NOTE: I'm not a lawyer, so take this advice as it is, and check with your attorney before taking any important business decision. I will not take responsibility for any business or financial loss, application rejection or removal that will arise, directly or indirectly, for following these guidelines. Use this information at your own risk.

The advice in this article applies mostly to consulting firms and independent consultants; obviously, if you develop your own products, you are the only one to hold the responsibility (and the only one to reap the benefits). Nevertheless, in the case of consulting, or when developing custom iPhone applications, you should shield yourself from the risk represented by Apple – **and yes, I mean and stress the use of the word "risk" here.**

In concrete terms, whatever your application does, there is a certain (relative) risk of being rejected or removed from the App Store by Apple, with or without valid reasons, in almost a random fashion. Being pessimist from the very beginning leaves you with the possibility of having a nice surprise later! If you work in consulting contracts, you **must** know, evaluate, and take this risk into account while doing estimations and contract negociations; the basis of this reasoning is that if your clients' applications have a certain degree of risk of being rejected, **you must not be held liable in that case.** Finally, you must do whatever you can to avoid your application from being rejected; this is your obligation, and the only means you have to increase your chances of selling your applications through the App Store.

iPhone software development is a highly risky activity[14]: consistent project management practices[15], human resource management[16], prototyping[17], education[18], quality management[19], memory management[20], code organization[21], keeping an eye on compiler warnings[22],

[11] http://bluewoki.com/

[12] http://rooifonts.com/

[13] http://web.me.com/bossofficer/iPhoneBioTech/Blog/Entries/2009/8/1_Apple__The_Unmanageable_Risk.html

[14] /blog/dirty-little-secret/

[15] /blog/scrum-software-development-process/

[16] /blog/adding-manpower/

[17] /blog/dangers-of-prototypin/

[18] /blog/challenges-for-software-engineers/

[19] /blog/software-project-quality/

[20] /blog/10-iphone-memory-management-tips/

[21] /blog/code-organization-in-xcode-projects/

[22] /blog/objective-c-compiler-warnings/

all these factors help reduce the risk of having your project enter the dreadful CHAOS statistics[23]. However, Apple's own politics regarding the App Store introduce a new level of risk into your project, unforeseen and, unfortunately, hard (but not impossible) to manage.

Explain the risks to the customer

Your customer must know about the App Store review process and all of its quirks. This is a reality. She might not need all the details, because more often than not, customers want an iPhone application just because "they have to be there", without caring that much about the latest geek details about the App Store rejection policies. But they must at least know the following facts:

1. **Depending on its features, an application might not be accepted in the App Store at all**; some reasons for rejection are known, but the most important thing to know is that "Apple reserves the right, in its sole discretion, to reject an application for any reason[24]", including, but not limited to, the (bad) mood of the reviewer[25].
2. Moreover, even if it has been approved in the App Store, and even if it has been already sold to some customers, **Apple reserves the right, in its sole discretion, to remove an application from the App Store at any time.** Boom[26].

As incredible as they might sound, your customer must know these facts, and you must keep a written track stating that your client acknowledges these facts, before the project starts. **It's their risk, not yours**; if once they acknowledge all of these factors, they still want their iPhone application, they will have to take care of this risk, and you shouldn't be taken as responsible for any problem caused by Apple.

I can't stress this enough: as always, keep a written track of the acknowledgement of these facts by the client, stating explicitly that you, as a developer, cannot be held responsible by arbitrary rejections not based in explicit facts. This means that if the rejection is due to non-respect of the UI guidelines, then yes, you must solve the problem as part of the warranty; but if the rejection is due to non-specified reasons, then it's not your responsibility at all.

Reading a draft of this article, Stephan Burlot[27] made an interesting comment about this particular point: if you go to your customer and tell him, just like that, "your app might never see the light of the App Store", you can be pretty sure he'll just run away, and hire another

[23] http://www1.standishgroup.com/newsroom/chaos_2009.php

[24] http://www.theiphoneblog.com/2009/06/12/apples-latest-app-store-rejection-policy/

[25] http://daringfireball.net/2009/05/diary_of_an_app_store_reviewer

[26] http://www.youtube.com/watch?v=r8L39UwOS-Y

[27] http://blog.coriolis.ch/

developer who will keep some of this truth away from his eyes, at least to get the contract. So the important thing here is to be professional and keep an eye on the most obvious factors that make Apple angry:

- Telephony services
- Porn and erotic content
- Exchange of music
- Access to the local music libraries of the device

Release dates cannot be guaranteed

Even if Apple acknowledges[28] that 9 out of 10 applications are nowadays approved in about two weeks (the incubation time is apparently increasing[29]), you can never be sure of the date when you will receive the nice "your app is ready for sale" e-mail[30].

Remember Schrödinger's cat[31]? Well, there's a bit of randomness in the date in which your application will be approved (if that ever happens at all), which means that **you must not commit to a defined application release date in your development contract**. Hence, when submitting your app, select a date at least 1 or 1.5 months in the future, and begin your marketing efforts as soon as the application is "ready for sale" in iTunes Connect. You can change the release date once your application has been approved.

Reduce the chances for rejection

Closely follow Apple's own iPhone Human Interface guidelines[32] for your applications, use some common iPhone UI patterns[33], avoid known UI design mistakes[34], and finally check out the App Rejected[35] and Application Submission Feedback[36] sites, and use the information as checklists for your applications, in order to verify as much as possible before submitting them to the App Store. **Make this a part of your quality management process, at the very end of the development cycle.**

Apple also published[37] some interesting tips last week, which can be

[28] http://developer.apple.com/iphone/news/

[29] http://radar.oreilly.com/2009/07/itunes-app-store-incubation-period-increases.html

[30] http://twitter.com/akosma/status/2946538079

[31] http://en.wikipedia.org/wiki/Schrödinger\XeTeXglyph\numexpr\XeTeXcharglyph"0027\relax{}s_cat

[32] http://developer.apple.com/iphone/library/documentation/userexperience/conceptual/mobilehig/Introduction/Introduction.html

[33] http://flyosity.com/application-design/iphone-application-design-patterns.php

[34] http://www.smashingmagazine.com/2009/07/21/iphone-apps-design-mistakes-overblown-visuals/

[35] http://www.apprejected.com/

[36] http://appreview.tumblr.com/

[37] http://developer.apple.com/iphone/news/#tipsonsubmittingyourapp

considered as “official” guidelines, even if I think that the information in the App Rejected and App Submission Feedback sites is (at the time of this writing) incomparably better.

Conclusion

To summarize: **I (like many others) still believe in this platform**. And I believe Apple will slowly remove some of the cruft from all of the process and make it more transparent.

Moreover, the risk exists, that Apple’s review team stands between you and the App Store; but even so, this risk is highly relative and manageable: at the time of this writing, there are over 63’000 apps already approved, and in July 2009 alone, 7’600 applications have been approved[38]! So chances are, many, many apps will still be approved, and yours has a large, large chance of getting into the party soon.

Interestingly, in June 2009, 9’887 apps have been approved, while 10’203 have been submitted[39]; this means that only 3% of the applications submitted were rejected… that leaves a 97% of approved applications! Even more important: **the rate of rejected apps seems to have a negative slope (both in absolute and relative figures)!**

Month	Submitted	Approved	Rejected	% Rejected
June 2008	19	17	2	10.53%
July 2008	597	527	70	11.73%
August 2008	1630	1482	148	9.08%
September 2008	2952	2626	326	11.04%
October 2008	2671	2343	328	12.28%
November 2008	3155	2822	333	10.55%
December 2008	3849	3483	366	9.51%
January 2009	4978	4445	533	10.71%
February 2009	6040	5504	536	8.87%
March 2009	7586	7052	534	7.04%
April 2009	8385	7839	546	6.51%
May 2009	8503	8123	380	4.47%
June 2009	10203	9887	316	3.10%

(Data derived from 148apps.biz[40]: app submissions[41] and total app count[42])

[38] http://148apps.biz/app-store-metrics/?mpage=appcount
[39] http://148apps.biz/app-store-metrics/?mpage=submission
[40] http://148apps.biz/
[41] http://148apps.biz/app-store-metrics/?mpage=submission
[42] http://148apps.biz/app-store-metrics/?mpage=appcount

Hope this helps! As I said above, I am not a lawyer, so all of this information is based in my own experience, working as an iPhone development consultant for over a year now. Feel free to add your own experiences in the comments section below, so that all of us can benefit from it.

Acknowledgements

Thanks to Stephan Burlot[43] for reading and commenting drafts of this article!

Update, 2009-08-03: Don't miss this brilliant MacWorld article[44] about the current evolution of the issue.

Update, 2009-08-04: Just found some more tips to avoid application rejection[45] (including a second part[46]).

Update, 2009-11-28: A new site is tracking down application rejections: http://apprejections.com/ (apparently http://www.apprejected.com/ does not work anymore).

[43] http://blog.coriolis.ch/
[44] http://www.macworld.com/article/142022/2009/07/appstore_innovation.html
[45] http://www.mobileorchard.com/avoiding-iphone-app-rejection-from-apple/
[46] http://www.mobileorchard.com/avoiding-iphone-app-rejection-part-2/

Discovering a Hidden iPhone URL Scheme

Adrian Kosmaczewski

2009-08-04

As an iPhone developer, one of the simplest and easiest mechanisms you have to interact with other applications is through the use of iPhone URL Schemes. These are so important that I've created a wiki page where I keep track of those I come across, including code samples that help me exchange data with them.

However, not all editors document the URL schemes they support in their apps, and this blocks reuse and collaboration. I recently came into such a problem, trying to use TwitterFon from my own apps, to post messages to Twitter. The TwitterFon site only specifies the following iPhone URL scheme:

twitterfon:///post?this%20is%20a%20test

The problem is, this URL scheme does not perform an URL-decoding on the message parameter, which means that a phrase like "this is a test" will appear in TwitterFon URL-encoded, that is, as "this%20is%20a%20test". Clearly not acceptable.

However, thanks to Ashley Mills, I learnt that the USA Today iPhone app is able to use TwitterFon to share articles via Twitter, and does this properly, without URL-encoded characters. How do they do that? Obviously, they are using an URL scheme exported by TwitterFon, but not documented anywhere (*). I finally discovered that the URL scheme sought is the following ("message" instead of "post"!):

twitterfon:///message?some%20text%20here

This is how I found out: I impersonated TwitterFon in my own iPhone with an ad-hoc app created in Xcode, that shows me the URL used by USA Today to launch TwitterFon.

These are the steps required:

- Open iTunes and look for the application whose URL schemes you're interested in (in my case, TwitterFon Pro); right click on it and select "Show in Finder";
- Duplicate the .ipa file in the Finder and change its extension to .zip
  - yes, .ipa applications are simply compressed .zip files;

- Uncompress the .zip file and open the folder; inside, navigate to the “Payload” folder, and right-click on the .app file inside; select “Show Package Contents”;
- Browse inside the package and open the Info.plist file; inside, you’ll find two keys that are interesting for us: **CFBundleURLTypes** (“URL types”) and **CFBundleIdentifier** (“Bundle identifier”). Select them and copy them to your clipboard;
- Create a new Xcode application, using the “iPhone OS / View-based application” template;
- Open the Info.plist file corresponding to the default target and remove the existing CFBundleIdentifier (“Bundle identifier”) key; paste the two items you’ve copied in the previous step - this means we’re creating an application that will “impersonate” itself as the “real” one;
- Modify the new Xcode project’s app delegate adding the following method:

```
- (BOOL)application:(UIApplication *)application
      handleOpenURL:(NSURL *)url
{
    viewController.viewer.text = [url absoluteString];
    return YES;
}
```

- Add a UITextView to your viewController (you can do this easily in Interface Builder, editing the .xib file), and expose it through a public property (called “viewer” above) so that the app delegate can access it;
- Prepare your Xcode project for ad-hoc deployment: add a “distribution” configuration, an entitlements.plist file, etc;
- Plug your iPhone, select “Distribution / iPhone OS Device”, and “Run” your application; the application will build and Xcode will install it into your device. **ATTENTION: this application will overwrite the original one!** This is because it has the same bundle identifier. Be sure to backup your data before doing this, as it will be lost completely;
- Now run the application calling the one you impersonate (in this case, the USA Today one) and force it to call the “impersonated” application (in this case, by “sharing” an article via Twitter). This will trigger the launch of the impersonated application, the call to application:handleOpenURL:, which itself will display the calling URL on the iPhone screen.

You can download a zip file with the Xcode project[1] created above if you want. Be careful if you run it on your own device!

Voilà! Finally, delete your own impersonated app, go to the App Store, re-install the application you’ve impersonated (normally it’s a free download, even for non-free apps), and you are done. The same mechanism could be used to find out similar, hidden URL mechanisms

[1] Fake.zip

in other apps.

(*) Actually this URL scheme is only shortly mentioned in the changelog of the application…

bluewoki Press Release

Adrian Kosmaczewski

2009-08-05

> akosma software is pleased to announce bluewoki 1.0, a simple and fun Bluetooth peer-to-peer walkie-talkie for iPhone OS 3.0, for iPhone and iPod Touch (2nd generation) with source code available on Github. bluewoki is a fun application which allows two iPhones to communicate over an ad-hoc Bluetooth network. It is a fun application to use with friends or relatives over a small distance, for example while biking or in a crowd.

Press release on prmac.com[1] and iPhone News Tracker[2].

[1] http://prmac.com/release-id-6801.htm

[2] http://www.iphonenewstracker.com/2009/08/05/bluewoki-1-0-released-bluetooth-walkie-talkie-for-iphone-os/

bluewoki on ITWire.com

Adrian Kosmaczewski

2009-08-06

Walkie-talkie program hits App Store

Article by Stephen Withers published on ITWire.com[1].

[1] http://www.itwire.com/content/view/26765/1231/

New iPhone Apps: RooiFonts and DeviceDNA

Adrian Kosmaczewski

2009-08-07

Let me introduce to you **RooiFonts** and **DeviceDNA**, the latest iPhone apps by akosma software[1] on the App Store!

RooiFonts[2] is an evolution of my previous Font Browser application[3] (still open source, still in Github[4]). RooiFonts builds upon that application bringing some more new features, like the ability to send a screenshot of a sample of text in the selected font via e-mail, or being able to compare two fonts side by side. RooiFonts is available in the App Store[5] for USD 3.99 (CHF 4.40, EUR 2.99).

[1] http://akosma.com/
[2] http://rooifonts.com/
[3] /blog/iphone-font-browser/
[4] http://github.com/akosma/fontbrowser/
[5] http://itunes.com/apps/rooifonts

On the other hand, DeviceDNA[6] is a free application for all of my clients, to send me their iPhone device information (including their UDID) via e-mail in a convenient way. No more explaining "open iTunes, click here, paste there…", just install this, and you're done.

As usual, both are available in English, French and Spanish.

[6] http://itunes.com/apps/devicedna

RooiFonts Press Release

Adrian Kosmaczewski

2009-08-09

Press release on prmac.com[1].

> akosma software is pleased to announce RooiFonts 1.0, a simple iPhone OS 3.0 font management tool for iPhone application designers and developers. RooiFonts is a tool for iPhone application designers, helping them to choose the right font for future iPhone applications. It allows them to study in detail, compare similar fonts, and share information with peers about the fonts available in the iPhone OS.

[1] http://prmac.com/release-id-6886.htm

Muchas Notitas, Muchas!

Adrian Kosmaczewski

2009-08-11

Notitas[1] is available at an App Store near you[2]! Notitas means "small notes" in Spanish, and it's the fourth iPhone application under the akosma[3] brand, and the first based on an original idea of my dear wife[4]!

As the name implies, it's a simple and easy way to create, keep and find notes in your iPhone, with some bonus: geographical awareness, so that each note remembers where it was created; the ability to publish notes in Twitter (for the moment, only if you have Twitterriffic installed in your iPhone) and send them via e-mail, too. I'm particularly happy of the Twitter integration (which prompted a whole article in this blog[5]) so that I can use Notitas a lot as a "draft Twitter" client.

I'm already preparing version 1.1 with TwitterFon + Tweetie support, as well as a German localization thanks to Sophie from Sophiestication[6]! Stay tuned for more goodies :)

PS: oh, and while you're using Notitas, try shaking the board a bit and

[1] http://muchasnotitas.com/
[2] http://itunes.com/apps/notitas
[3] http://akosma.com/
[4] http://twitter.com/claukosma
[5] /blog/discovering-a-hidden-iphone-url-scheme/
[6] http://www.sophiestication.com/

you’ll see what happens ;)

Notitas Press Release

Adrian Kosmaczewski

2009-08-19

Press release on prmac.com[1], Breaking Windows[2], TechWhack[3], MacCrazy[4], AppPodcast[5], 148apps[6], WPZines[7], MacTrack[8], MacMegasite[9], Desinformado[10], iFones[11], AppleUsers.org[12], MacOSXNews[13], Charged[14].

> akosma software today is pleased to announce Notitas 1.0 (“little notes” in Spanish), a geographically-aware virtual board where you can post text notes of any kind, change their font, their color, share them via e-mail and Twitter, and view the exact location where you created them.

[1] http://prmac.com/release-id-7067.htm

[2] http://breakingwindows.com/prmac/standalone.php?prmac_id=7067

[3] http://press-releases.techwhack.com/39922-notitas

[4] http://www.maccrazy.net/2009/08/19/notitas-10-released-simple-and-funny-notes-for-iphone-os/

[5] http://theapppodcast.com/notitas-1-0-released-simple-and-funny-notes-for-iphone-os

[6] http://148apps.biz/notitas-1-0-released-simple-and-funny-notes-for-iphone-os/

[7] http://wpzines.com/uncategorized/2009/08/19/notitas-10-released-simple-and-funny-notes-for-iphone-os.html

[8] http://themactrack.com/2009/08/19/notitas-10-released-simple-and-funny-notes-for-iphone-os/

[9] http://macmegasite.com/node/7703

[10] http://iphonenews.desinformado.com/2009/08/notitas-10-released-simple-and-funny-notes-for-iphone-os/

[11] http://ifones.com/notitas-10-released-simple-and-funny-notes-for-iphone-os/

[12] http://news.appleusers.org/prmac/notitas-1-0-released-simple-and-funny-notes-for-iphone-os/

[13] http://macosxnews.com/5938

[14] http://www.charged.co.za/apple-world-events/notitas-10-released-simple-and-funny-notes-for-iphone-os

Speaking in Copenhagen and Zürich

Adrian Kosmaczewski

2009-08-26

Just a quick post to tell you all that I'll be speaking tomorrow evening (August 27th, 2009) in Copenhagen (Denmark) in the free JAOO Geek Night[1] organized by the great team of Trifork[2]. They are the organizers of world-class events such as the QCon London[3], QCon SF[4] and RubyFoo London[5]!

But even better news for the Swiss: the same event will happen in Zürich, on September 9th[6] so feel free to come, have a snack with us, share a few drinks and talk about iPhone development.

See you there!

[1] http://url.akosma.com/3jlr3h
[2] http://www.trifork.com/
[3] http://qconlondon.com/
[4] http://qconsf.com/
[5] http://jaoo.dk/ruby-london-2009/
[6] http://url.akosma.com/g04dq7

Mention on Christer Østergaard's Blog

Adrian Kosmaczewski

2009-08-28

A talk on iPhone development[1]" in Christer Østergaard's blog[2].

> Had the pleasure of seeing Adrian Kosmaczewski perform an excellent and enthusiastic presentation about iPhone development Thursday evening. Although I'm a senior developer on the Microsoft .Net platform, I'm a complete iPhone novice. It was therefore nice to attend this sort of "brain dump"-session, where we were given a brief walkthrough of the history of Objective-C, introduction to the development environment and then got into some example code.

[1] http://www.christer.dk/post/A-talk-on-iPhone-development.aspx
[2] http://www.christer.dk/

Article on SonntagsZeitung

Adrian Kosmaczewski

2009-08-30

> Der Programmierer Adrian Kosmaczewski aus Lausanne arbeitet bereits seit knapp drei Monaten mit einer Vorabversion des Betriebssystems. Sein Urteil fällt eindeutig aus: “Ich bin sehr beeindruckt und empfehle dieses Upgrade jedem Mac-Nutzer”.

“Flinker Schneeleopard”, article published on the SonntagsZeitung[1] (download PDF scan of the article[2]).

[1] http://www.sonntagszeitung.ch/

[2] /press/SonntagsZeitung-Snow-Leopard.pdf

Copenhagen Slides Available

Adrian Kosmaczewski

2009-08-31

Another quick post to announce that I've published the slides (and sample application) I've shown last week in Copenhagen; feel free to download them from the projects section[1]!

Remember that I'll be hosting a similar talk on September 9th in Zürich[2]! I look forward to see you there and to discuss about iPhone development.

[1] /projects/tips-and-tricks-copenhagen/
[2] http://url.akosma.com/g04dq7

Slides, slides, slides

Adrian Kosmaczewski

2009-09-10

I've been doing presentations for a while now, so I decided to open a SlideShare account[1] to publish all the slides I've created over the past 5 years. SlideShare has a great Flash-based viewer that you can embed in web pages, so I'll be using it a lot now. Check out my presentations, feel free to download them and also to use them if you find the contents useful for you (they are distributed with Creative Commons licenses).

Having said that, I'm also announcing that the slides (and sample application) of yesterday's JAOO geek night presentation in Zürich are also available in the Projects section of this blog[2], and here goes the SlideShare player with those slides:

Update, 2022-09-09: Looking backward, I realize haven't used SlideShare that much; around 2011 I moved to Speaker Deck and that's where[3] you can still find my presentation slides. I'm even a paying customer of theirs.

[1] http://www.slideshare.net/akosma
[2] jaoo_presentation.zip
[3] https://speakerdeck.com/akosma

Using Multiple Twitter Clients from your iPhone Application

Adrian Kosmaczewski

2009-09-11

I love playing with iPhone URL schemes[1]. And if you ask me, just like for Mail.app and Safari, I think there should be a "default" Twitter[2] client URL scheme in the iPhone, with an interface in the Settings application allowing you to set the application that you prefer to tweet. Alas, this is not the case, and each application must manually allow users to select their preferred client, from a list of known ones.

Having documented iPhone URL schemes[3] for TwitterFon[4], Twitterrific[5], Tweetie[6] and Twittelator[7], I've created a project, available in Github, called TwitThis, which helps users choose their preferred Twitter client, and makes it easy to remember the user choice, and to launch the associated application with a single command:

This application has the following features:

- The class TwitterClientManager loads a list list of supported Twitter clients is loaded from a plist file, which can be extended to support more clients in the future;
- Each Twitter client is represented by an instance of the TwitterClient class;
- The user can choose his preferred Twitter client at any time, and launch the application by a simple touch; the TwitterClientManager class stores the selected value in the user settings.

If you have to support several different Twitter clients, feel free to use these classes in your own project! The project, as usual, is available in Github[8] with a liberal BSD license. Enjoy! I'd love to hear your comments below.

---

[1] /blog/discovering-a-hidden-iphone-url-scheme/
[2] http://twitter.com/
[3] http://wiki.akosma.com/IPhone_URL_Schemes
[4] http://twitterfon.net/
[5] http://iconfactory.com/software/twitterrific
[6] http://www.atebits.com/tweetie-iphone/
[7] http://www.stone.com/Twittelator/
[8] http://github.com/akosma/TwitThis/

Epic Interview: A New Literary Genre in the Tech Section?

Adrian Kosmaczewski

2009-09-13

Here's a simple recipe:

1. Contact the most important people in some field.
2. Sit down and ask a similar set of questions to each one of them.
3. Record all the interviews and then write them down.
4. Publish the resulting book, usually with great reviews (such as this one).

This does not constitute, by any means, a new genre; but it's certainly a fashionable one in your technical bookstore right now. At least Apress[1] and O'Reilly[2] have realized that this simple technique yields epic books.

I have already blogged about Founders at Work[3], thus it's worth mentioning that Coders at Work[4] (which I'm reading right now) has just been released. Both books share a similar structure (as well as a similar cover), and both are highly recommendable, with interviews[5] of David Heinemeier Hansson, Steve Wozniak and Paul Buchheit for the first, and Donald Knuth, Joe Armstrong and Brendan Eich for the second.

On the other side, O'Reilly is very well aware of the force conveyed by this kind of books: their "/Theory/In/Practice" series of books[6] has some gems which, I think, are really worth reading.

"Beautiful Code" features interviews with Brian Kernighan, Charles Petzold and Yukihiro Matsumoto; "Beautiful Teams" (already my preferred book for 2009!) features Tim O'Reilly, Barry Boehm and Grady Booch; finally, "Masterminds" has great interviews with Bertrand Meyer, Bjarne Stroustrup, James Gosling, Brad Cox and Anders Hejlsberg.

[1] http://apress.com/
[2] http://oreilly.com/
[3] /blog/best-books-of-2007/
[4] http://www.codersatwork.com/
[5] http://www.foundersatwork.com/interviews.html
[6] http://oreilly.com/store/series/theory.html

I think that the names of the interviewees, in each of the five books, speak for themselves. In all of them, I have found inspiration, advice, tips, humour, awe and enlightenment. The good thing being that, in most cases, you don't need a Computer Science degree to read these books; it's just a matter of empathy and sociology. Our world is driven by software, and the stories behind its construction are not only interesting, they are also important to understand the cost, the difficulty and the wonder that constitutes a piece of working software. These books are a way to approach the immense complexity of our society.

I really look forward to read more books of this kind! If I forgot to mention any other similar book, just leave the reference in the comments section below. I'd love to read your suggestions.

Lift Conference iPhone Application

Adrian Kosmaczewski

2009-09-25

"Lift on your iPhone"[1], article by Laurent Haug[2] in the Lift Conference website[3].

> Lift is coming to everybody's favorite phone, one day after the iPhone was finally announced in Korea! Here comes our iPhone app on which you can watch our talks, push them to your people, bookmark your favorites, and find out about our upcoming events.
>
> The application was developed in partnership with Akosma software and our video guru Thierry Weber who tell me that if your organization has online videos it wants to spread on the iPhone you should contact them!

[1] http://liftconference.com/news/lift-your-iphone
[2] http://liftconference.com/person/laurent-haug
[3] http://liftconference.com/

Review de Notitas sur L'Express.fr

Adrian Kosmaczewski

2009-09-30

"Notitas, IM+, Old Booth Premium"[1] par Tardis Girl[2]; review de Notitas[3] sur L'Express.fr[4]

> Présenté sous la forme d'un panneau de liège, Notitas vous permet de prendre des notes et de les punaiser sur un mur virtuel. Le tout de façon ultra simple. Bien pensé et graphiquement très sympa, le programme permet aussi d'envoyer les notes rédigées via email ou via Twitter si on possède le programme Twitterriffic. De bonnes idées pour une application simple, mais efficace.

[1] http://www.lexpress.fr/actualite/high-tech/notitas-im-plus-old-booth-premium_791232.html

[2] http://inthetardis.net/

[3] http://muchasnotitas.com/

[4] http://www.lexpress.fr/

iPhone and Mac OS X Developer Conference Roundup

Adrian Kosmaczewski

2009-10-01

Here's a quick review of the most important iPhone and Mac OS X developer conferences I've found on the web (in no particular order). Definitely, there's no shortage of conferences when you need information about the latest Cocoa, Mac OS X and iPhone technologies; check this out!

- Apple's Worldwide Developer Conference[1] or WWDC, held every year in San Francisco, CA (USA), usually around June, and featuring presentations from Apple employees; if you've never been to one, believe me, you should;
- Voices That Matter iPhone Developer Conference[2] to be held in Boston, MA (USA) next October 17th and 18th, with (among others) Erica Sadun, Aaron Hillegass, Stephen Kochan and Marcus Zarra;
- NSConference[3], to be held from January 31st to February 3rd next year near Reading (UK), and from February 21st to February 24th in the Georgia Tech Institute, GA (USA), featuring (among others) Matt Gemmell, Marcus Zarra and Aaron Hillegass;
- The 360|iDev conference[4] that just finished in Denver, CO (USA), which featured (among many others) Bill Dudney, Brent Simmons, and Marcus Zarra (definitely, Marcus Zarra is everywhere!);
- The iPhone Developer Summit[5] in Santa Clara, CA (USA) next November 3rd;
- The iGames Summit[6], a conference targeted to iPhone game developers, held last March in San Francisco, CA (USA), featuring (among many others) Neil Young (from ngmoco), Andrew Lacy (from Tapulous) and Mike Mettler (from AdMob);
- The Macoun Entwicklerkonferenz[7] which happened last September 26th in Frankfurt (Germany);

[1] http://developer.apple.com/wwdc/
[2] http://www.voicesthatmatter.com/iPhone2009/
[3] http://www.nsconference.com/
[4] http://www.360idev.com/
[5] http://www.iphonedevsummit.com/
[6] http://www.igsummit.com/
[7] http://macoun.de/

- The iPhone developer conference[8] in Köln (Germany), in December 1st and 2nd;
- And finally, the JAOO iPhone Dev Day[9] in Zürich (Switzerland) next October 8th, featuring Raven Zachary, Alex Cone and... many others ;)

Also noteworthy, but not so much about software development I think, is the Mobile Enterprise Conference[10] in Amsterdam (Netherlands) on November 3rd, which has a couple of tracks about the iPhone in enterprise.

Feel free to add links to other similar events elsewhere in the world!

Update, 2009-10-02: Here's the link to Jonathan 'Wolf' Rentzsch's C4 Independent Developers Conference[11].

Update, 2009-10-03: The Øredev 2009 Developer Conference[12] in Malmö (Sweden) next November has iPhone / Mobile tracks too. And so will the Scandinavian Developer Conference 2010[13] in Göteborg (also in Sweden).

[8] http://iphonedevcon.de/
[9] http://iphonedevday.com/
[10] http://www.mobileenterprise09.com/
[11] http://rentzsch.com/c4
[12] http://www.oredev.org/
[13] http://scandevconf.se/

About the JAOO Conferences

Adrian Kosmaczewski

2009-10-05

This week I had the opportunity to attend the JAOO Developer Conference 2009[1] in Århus (Denmark), invited by Trifork[2], the company behind this and other fine events, like QCon[3] and RubyFoo[4]. Despite being relatively unknown in the Swiss landscape, JAOO is an event unlike any other, and here's why you should attend next time.

Trifork started organizing JAOO conferences around 1996. At that time, Java was the hottest thing on the programming landscape, and Trifork thought (rightly so) that Java-oriented conferences could be a success. Over the years, JAOO evolved to encompass many other subjects, like Inversion of Control, Design Patterns, Architecture, Open Source, Functional Programming, and of course every possible trend the industry has enjoyed (or suffered) in the past 13 years. In some cases, like the RubyFoo event in London[5] new events have spawned from JAOO to respond to growing new trends.

The first distinctive fact about JAOO is, then, its diversity and agnosticism. You are more likely to find a speaker about your favourite technology or programming language here than in any other conference, except perhaps lately in the StackOverflow DevDays[6], which share with JAOO the openness and breadth, if not the maturity. The advantage of such "mixed" conferences is the ability to contrast approaches and discuss alternatives, something usually more difficult in

[1] http://jaoo.dk/aarhus-2009/
[2] http://www.trifork.com/
[3] http://qcon.infoq.com/
[4] http://jaoo.dk/ruby-london-2009/
[5] http://jaoo.dk/ruby-london-2009/
[6] http://stackoverflow.carsonified.com/

conferences like Apple's WWDC[7] or Microsoft's TechDays[8], given the evident bias these have.

Another unique element of JAOO is the list of speakers. Take a peek on the current[9] and past speakers[10] who ever gave a speech in JAOO: Barry Boehm, Yukihiro Matsumoto, Martin Fowler, Charles Simonyi, and I can't name them all without blinking my eyes and swearing for not having attended JAOO before. The list is simply a "who is who" of software engineering.

There is another element that makes JAOO stand among developer conferences, and it's the commitment of trying to improve not only our minds, but also our bodies: take a look at the JAOO IT Run[11], a nice and original response to the (real) problem of overweight in the IT industry. And it sure is a success, even if I'm not sure I would sustain more than a kilometer without a heart transplant.

Finally, JAOO also has an interesting social commitment, trying to bridge the huge gap in the number of men and women working in this industry: at JAOO conferences, every attendee has the right to invite another person of the opposite sex![12] This initiative is not only great, it's a positive step to make women join the ranks of software engineers worldwide, and I think it's an idea that should be emulated elsewhere.

In any case, I was not only surprised by JAOO's quality, breadth, topics, interestingness, but also by its social and human side, trying to adopt initiatives that make us not only better engineers, but also better human beings, in a better society.

Disclaimer: I'm a speaker on theJAOO iPhone Dev Days Zürich 2009[13] next Thursday.

[7] http://developer.apple.com/WWDC/
[8] http://www.microsoft.com/switzerland/msdn/de/techdays/
[9] http://jaoo.dk/aarhus-2009/speakers/
[10] http://jaoo.dk/archives/alltimespeakers/
[11] http://jaoo.dk/aarhus-2009/it-run/
[12] http://jaoo.dk/aarhus-2009/women/
[13] http://iphonedevday.com/suisse-2009/

Interview on the Tages Anzeiger

Adrian Kosmaczewski

2009-10-05

Wie man mit dem iPhone Geld verdient[1], interview by Roger Zedi published in the Tages Anzeiger[2] (download the PDF file with the printed article[3]).

> – Wie viel Erfahrung muss man mitbringen, wenn man eine Applikation schreiben möchte? – Man sollte schon einmal mit einer Programmiersprache wie C++ gearbeitet haben. Wer bisher JavaScript und Ähnliches benutzt hat, muss noch einiges dazulernen, umdenken. – Könnte das populäre Telefon auch Junge animieren, ins Programmieren einzusteigen? – Es gibt schon junge Talente, die direkt auf dieser Plattform einsteigen und Erfolg haben.

[1] http://www.tagesanzeiger.ch/digital/mobil/Wie-man-mit-dem-iPhone-Geld-verdient/story/22603890

[2] http://www.tagesanzeiger.ch/

[3] /press/TagesAnzeiger_interview.pdf

JAOO iPhone Dev Days 2009 Zürich

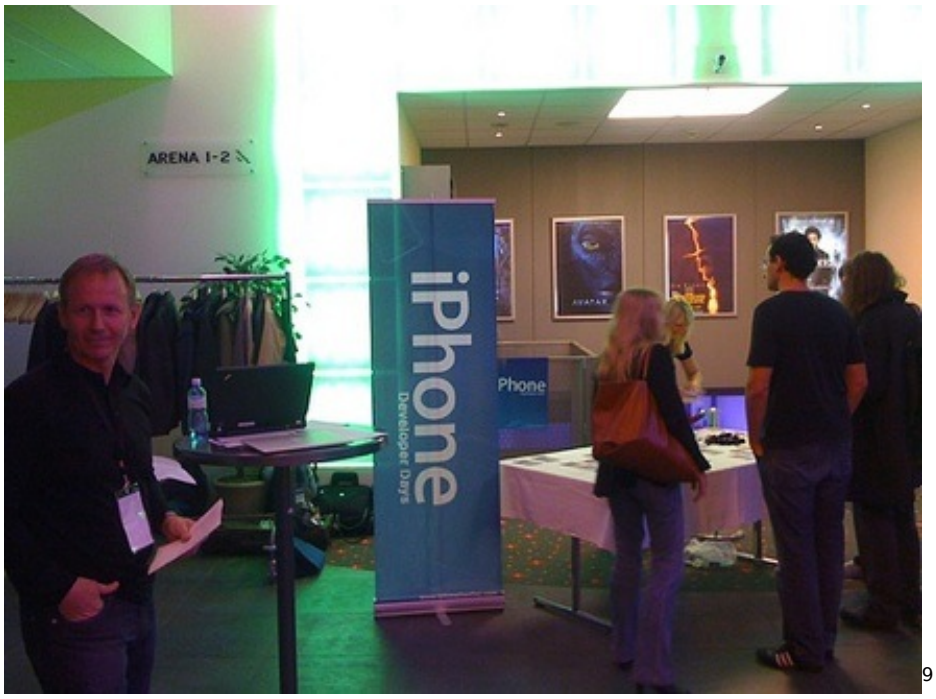
Adrian Kosmaczewski

2009-10-13

Last week's JAOO iPhone Dev Day[1] was a big success. Featuring Raven Zachary[2], Alex Cone[3], Jonas Schnelli[4], Patrick Bönzli[5] and Patrick Linskey[6] and yours truly, the event gathered many attendees interested in the capabilities of the iPhone for their businesses.

This is a small review of the event, organized by the incredible teams of Trifork[7] and Keynode[8] with links to the material I've provided in my own presentations.

[1] http://iphonedevday.com/suisse-2009/
[2] http://raven.me/
[3] http://www.linkedin.com/in/alexcone
[4] http://www.include7.ch/
[5] http://iphonedevday.com/suisse-2009/speaker/Patrick+Bönzli
[6] http://jaoo.dk/speaker/Patrick+Linskey
[7] http://trifork.com/
[8] http://www.keynode.biz/



[9]

Raven Zachary[10] needs no introduction. Known by the masses as the "Obama iPhone App[11] Guy", he's a brilliant entrepreneur who's jumped onto the iPhone wagon without restrictions. His company in Portland, Oregon, provides not only iPhone app development services, but also marketing and business information for companies in the field. His presentation was insightful, interesting and fun; Raven has a unique style for presenting data and he captured the audience with incredible information. His presentation provided a unique and broad perspective to the whole market, preparing the audience for the more technical presentations later.

[9] http://www.flickr.com/photos/akosma/3991712927/
[10] http://raven.me/
[11] http://my.barackobama.com/page/content/iphone

[12]

Jonas Schnelli is a dear friend of mine, founder and CEO of Include7[13]. He followed Raven on stage, talking about the all-time famous SBB iPhone Application[14]. Jonas talked about how he came up with the idea (as you can imagine, he solved his own travel needs, which is the best way to start a project anyway) and how he developed a business relationship with the Swiss train company. He also provided 5 fundamental tips about iPhone development, coming from an experienced iPhone developer, thus closing a brilliant session with lots of interesting details.

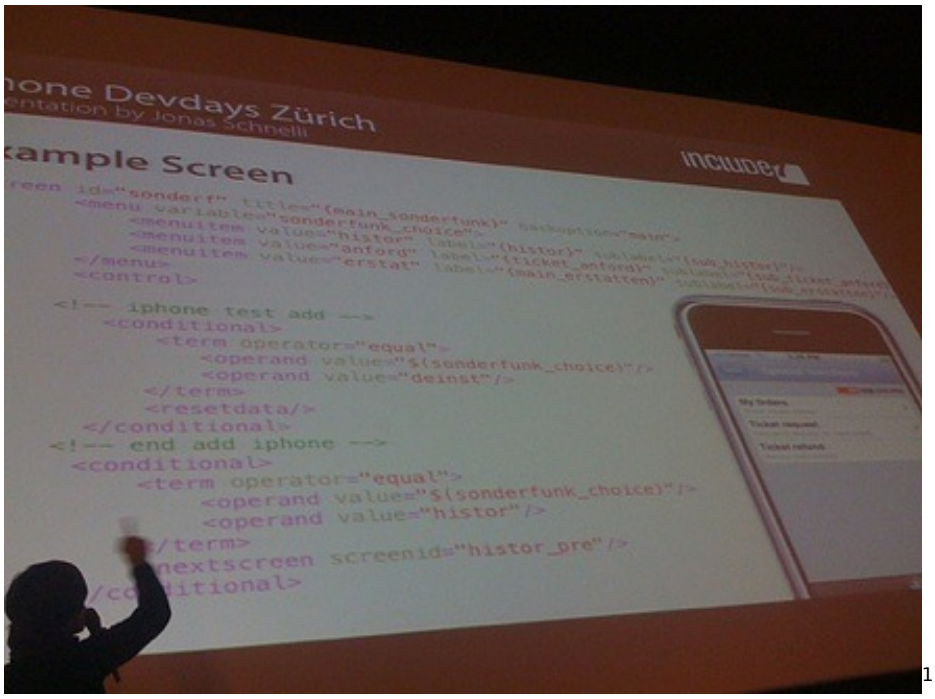
[12] http://www.flickr.com/photos/akosma/3992510002/
[13] http://www.include7.ch/
[14] http://mct.sbb.ch/mct/reisezeit/news/sbb-mobile-iphone.htm

15

Jonas also showed a new open source project his company is creating, an XML-based DSL[16] used to generate complex user interface interactions on the iPhone, which certainly looks promising.

[15] http://www.flickr.com/photos/akosma/3991861189/

[16] https://projects.include7.ch/projects/show/dialoginterpreter


[17]

Alex Cone[18] is the founder of the iPhone Dev Camps[19], and he has been working with NeXT and Apple technologies for almost 20 years now: he knows Cocoa as if he had coded it himself. He has even worked in the iTunes Store team at Apple! He provided a thorough insight on the architecture of iPhone applications: code organization, the use of notifications versus delegation, and many other interesting subjects related to the creation and maintenance of big Cocoa Touch projects. All in all, an amazing presentation which closed an exciting morning.

[17] http://www.flickr.com/photos/akosma/3992660932/
[18] http://www.linkedin.com/in/alexcone
[19] http://www.iphonedevcamp.org/

[20]

His visions are extremely interesting, and not only because the word “architecture” sounds strange in the context of small mobile apps: software is software, and architecture is key for quality, maintainability and happy developers.

[20] http://www.flickr.com/photos/akosma/3992739002/

[21]

After lunch I had the pleasure of giving a talk named “10 Commandments for iPhone Software Development”, a tongue-in-cheek presentation about do’s and dont’s for creating long lasting, best-selling applications, both from a development and design points of view. I hope everyone enjoyed it! You can find below the slides of the presentation I used. Interestingly, this presentation has been featured last week on SlideShare[22] as one of the most popular ones on Twitter!

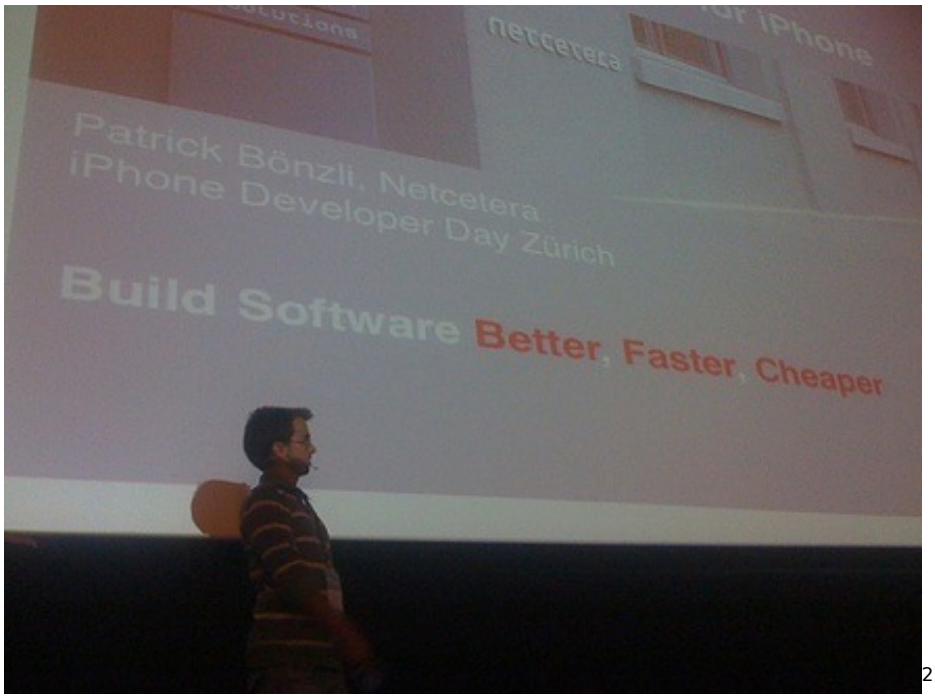
Right after came Patrick Linskey[23], who provided the audience with a first hands-on tutorial about how to build an iPhone application in 45 minutes. Great stuff, particularly given that most of the people in the audience were new to the platform, and his introduction showed how to add features using both Xcode and Interface Builder, highlighting the most important elements of the workflow.

[21] http://www.flickr.com/photos/akosma/3992776546/
[22] http://slideshare.net/
[23] http://jaoo.dk/speaker/Patrick+Linskey

[24]

Finally, Patrick Bönzli[25] from netcetera[26] talked about continuous integration on the iPhone.

[24] http://www.flickr.com/photos/akosma/3993039606/
[25] http://iphonedevday.com/suisse-2009/speaker/Patrick+Bönzli
[26] http://netcetera.ch/



[27]

Finally, Alex Cone and I split the audience in two, featuring two simultaneous hands-on tutorials on creating iPhone applications. Alex undertook the task of providing an excellent advanced tutorial, while I guided my audience into creating their first iPhone application. Unfortunately, the task took longer than expected, but you can find the final application, including the complete step-by-step log, in the Backlog project[28] in Github. Feel free to clone the project, explore the log, and see how an simple application can be created in just a couple of hours.

[27] http://www.flickr.com/photos/akosma/3993160390/
[28] http://github.com/akosma/Backlog

[29]

[29] http://www.flickr.com/photos/akosma/3992503310/

Roundup of Swiss Companies Writing Mac Apps

Adrian Kosmaczewski

2009-10-23

A lot has been said and done about the iPhone, but there's much more to Cocoa and Objective-C than our beloved pocket device.

During a chat session with Stefan Fürst from Media Atelier[1] we put together a quick list of significant Mac applications created in Switzerland (and southern Germany), and indeed the list (in no particular order) is nothing short of impressive; check it out:

- Zattoo[2], a really great TV player (cross-platform, actually, but the Mac version works really well);
- Cornerstone[3], one of the best Subversion clients for the Mac OS X operating system (the other being Versions[4]);
- TimeLog[5] and GrandTotal[6], the two applications used by independent contractors (like me) to keep track of the time spent in projects, and then to generate invoices out of that data – and which I personally prefer and strongly recommend over Billings[7];
- Distribute[8] by Seven Lakes Software, dubbed the best ERP software available in the Mac, with an impressive feature list and lots of positive reviews;

[1] http://mediaatelier.com/
[2] http://zattoo.com/
[3] http://www.zennaware.com/
[4] http://versionsapp.com/
[5] http://www.grandtotal.biz/TimeLog4/
[6] http://www.grandtotal.biz/GrandTotal/
[7] http://www.billingsapp.com/
[8] http://www.sevenlakessoftware.com/

- Snowtape[9], an application I've become addicted to lately, which allows you not only to listen to internet radio stations... but also to record them as MP3 or AAC files, which are automatically imported into iTunes!
- Waveboard[10], a Google Wave client for Mac (and iPhone soon, too!);
- Mailplane[11], a native Gmail clients with awesome capabilities;
- Posterino[12], Proxymind[13] and Snippet Mind[14] from Zykloid[15];
- Background Music[16] by infoAtelier, currently in a promising beta test phase;
- iBackup[17];
- GraphClick[18], iLocalize[19], ProVoc[20], XS[21] and AudioXplorer[22] by Arizona Software[23] (even if they aren't in Switzerland anymore I think, they are definitely worth including in this list);
- PowerMail[24] and FoxTrot[25] by CTM Development[26];
- LiquidCD[27] by Maconnect
- Several Mac products[28] by Stephan Burlot's Coriolis Technologies[29];
- Don't forget that most Cocoa apps are unit-tested with Sen:te[30]'s original OCUnit unit testing framework[31] bundled with Xcode since 2005!
- And last but not least, the diverse suite of Logitech drivers and control panels[32] written by Men in Silicium[33] in Geneva!

It really looks like the Swiss enjoy writing Mac apps (I certainly do and will publish mine soon!). Have I forgotten anyone? Please don't be upset, and feel free to leave your links in the comments below. I'd

[9] http://www.snowtape.com/
[10] http://www.getwaveboard.com/
[11] http://mailplaneapp.com/
[12] http://zykloid.com/posterino
[13] http://zykloid.com/proxymind
[14] http://zykloid.com/snippetmind
[15] http://zykloid.com/
[16] http://infoatelier.com/site/
[17] http://www.grapefruit.ch/iBackup/
[18] http://www.arizona-software.ch/graphclick/
[19] http://www.arizona-software.ch/ilocalize/
[20] http://www.arizona-software.ch/provoc/
[21] http://www.arizona-software.ch/xs/fr/?ref=en
[22] http://www.arizona-software.ch/audioxplorer/
[23] http://www.arizona-software.ch/
[24] http://www.foxtrot.ch/powermail/
[25] http://www.foxtrot.ch/foxtrot/
[26] http://www.foxtrot.ch/
[27] http://www.maconnect.ch/
[28] http://www.coriolis.ch/en/products.html
[29] http://www.coriolis.ch/
[30] http://www.sente.ch/
[31] http://www.sente.ch/s/?p=535&lang=en
[32] http://meninsilicium.com/fr/achievements.html
[33] http://meninsilicium.com/

love to know who else is creating killer apps for the Mac in a radius of 300 km around Zürich!

Disclaimer: I'm not affiliated with any of these companies (I'm just a friend of Stefan). And if you're more into "enterprisey" stuff, here's the "Swiss-Made Software" label site[34] that you migth find more interesting ☺

Update, 2009-10-23: Some more applications added after the publication of this post:

- Luscious SMS[35], the SMS client for the Mac;
- Special mention for Cyberduck[36], an open source FTP, SFTP, WebDAV, Cloud Files & Amazon S3 Browser for Mac OS X, or "the poor man's version of Transmit[37]"

Update, 2009-10-24: Sophie Teuschler[38] tells me not to forget the multiple Apple Design Award winners SubEthaEdit[39] and BoinxTV[40] by The Coding Monkeys[41], not far from Switzerland, in Bavaria…!

Update, 2009-10-25: I've just received an e-mail from Cyril Pavillard about his company Mnemis[42] and their product Uniboard[43] which looks absolutely awesome by any standards. Be sure to check out this cool Swiss project!

Update, 2009-11-12: Just found out about noidentity[44] and their MoneyBook iPhone application.

[34] http://swissmadesoftware.org/
[35] http://www.luscious-sms.net/
[36] http://cyberduck.ch/
[37] http://www.panic.com/transmit/
[38] http://www.sophiestication.com/
[39] http://www.codingmonkeys.de/subethaedit/index.html
[40] http://boinx.com/boinxtv/overview/
[41] http://www.codingmonkeys.de/
[42] http://www.mnemis.com/
[43] http://getuniboard.com/
[44] http://www.noidentity.ch/

On the Importance of Yerba Mate in the Software Development Process

Adrian Kosmaczewski

2009-10-26

This paper will highlight the results of an extensive research conducted since the mid 90's, on the effects of the consumption of beverages based in the plant known as Ilex paraguariensis, in the framework of software development process activities in South America and some small parts of Europe.

This paper will provide an introduction to the herb commonly referred to as "Yerba Mate", and will later delve into the advantages and disadvantages of such practice, in the context of the creation of software products.

Introduction

Yerba Mate is defined by Wikipedia[1] as follows:

> Yerba mate or yerba-mate (Br.) (Spanish: yerba mate, Portuguese: erva-mate), Ilex paraguariensis, is a species of holly (family Aquifoliaceae) native to subtropical South America in northeastern Argentina, eastern Paraguay and southern Brazil. It was first scientifically classified by Swiss botanist Moses Bertoni, who settled in Paraguay in 1895.

The Yerba Mate (usually and wrongly spelled as "Yerba Maté" in English-speaking texts) is used in the preparation of a caffeinated beverage described by Wikipedia[2] as follows:

> Mate (Spanish pronunciation: [mate]), also known as chimarrão or cimarrón, is a traditional South American infused drink. It is prepared from steeping dried leaves of yerba

[1] http://en.wikipedia.org/wiki/Yerba_maté
[2] http://en.wikipedia.org/wiki/Mate_(beverage)

> mate (Ilex paraguariensis, known in Portuguese as erva mate) in hot water. It is the national drink in Argentina, Paraguay, and Uruguay, and drinking it is a common social practice in parts of Brazil, Chile, eastern Bolivia, Lebanon and Syria. In Brazil, it is considered to be a tradition typical of the "Gaúchos", name given to those born in Rio Grande do Sul. The drink contains caffeine.
>
> (...)
>
> The multicultural Yerba Mate Association of the Americas states that it is always improper to accent the second syllable, since doing so confuses the word with the unrelated Spanish word meaning "I killed."

One of the phrases in the quoted paragraphs from Wikipedia brings to mind the importance of such a drink in the creation of software products (no, not the phrase about killing, the previous one). Caffeine is known for its capabilities in waking up inert areas of the brain, particularly during brain-damaging activities.

We consider unfortunate to qualify software development as a brain-damaging activity (although some research arrives to this particular conclusion), however, it is certainly a brain-intensive one, and as such, Yerba Mate has proven, in our tests, to be a particularly interesting option to coffee.

Preparation

To prepare "Mate" (the beverage), three basic elements are required:

1. A recipient, usually also referred to as "mate" (to add to the confusion), but also called "guampa", "cuia", "calabaza", and other names without any translation to English whatsoever. Among these names appears also "porongo", as it is known in Uruguay; this word is usually avoided in Argentina, for the exact same reason the name "Mitsubishi Pajero"[3] has been a commercial failure there. This element can be made of wood, metal or even be the hollow shell of a dried calabash[4].

[3] http://en.wikipedia.org/wiki/Mitsubishi_Pajero
[4] http://en.wikipedia.org/wiki/Calabash

2. A metallic straw, usually also referred to as “bombilla” or less commonly, “bomba”. This element can be made out of metal or wood, and is used to drink the infusion, avoiding to swallow the leaves of Yerba Mate at the same time. The best ones have their top part covered in gold, which protects the lips from the intense heat generated by the water in the metal, and also provides a sense of luxury into an otherwise rather humble activity.

3. Hot water, never boiled, at around 70 to 80 degrees Celsius (160 - 180 degrees Fahrenheit). It is very, very, **VERY** important to serve the water at the exact temperature, without boiling the water inadvertently. Usually, the best way to keep the water hot is with a thermos or vacuum flask[5], of which the latest industry benchmarks highlight the Uruguayan brand “Lumilagro”[6] as the most reliable, competitive and durable in the market. European customers are best served by the standard thermos provided by Ikea[7].

Once all the elements are ready, the preparation process is fairly simple:

1. Add the Yerba Mate leaves in the mate (the recipient);
2. Put the right hand on top of the mate (recipient) covering the entrance, and using your left hand, turn the recipient upside down and shake it a little; then return the recipient to its normal position and dust the mate powder from your hand (it is strongly recommended **not** sniffing it);
3. Insert the straw in the recipient, creating a small hole in the Yerba at the same time;
4. Pour in hot water, very slowly, in the hole caved in the previous step; on the first serve it is best to avoid filling the mate completely, to leave time to the yerba to get moist and release the flavor slowly;
5. Drink the mate, by sipping at the straw, taking care not to burn your mouth or throat;
6. Pass the mate around, which helps create and spread a sense of teamwork, to bring an ambience of relaxation and self-contemplation, and also to spread many known viruses.

[5] http://en.wikipedia.org/wiki/Vacuum_flask
[6] http://www.lumilagro.com/
[7] http://www.flickr.com/photos/gahjr2000/5796110/

Advantages

In the context of software engineering, such a practice has the following advantages:

- **Health benefits:** The ingestion of mate (the beverage) contributes positively to the recommended daily intake of water (at least around 2 or 3 liters a day), and thus to the maintenance of a convenient hydration level in the brain, which is recognized by several studies as a major contribution to its productivity. Some recent papers even indicate that the habit of Mate drinking can reduce the risks of cancer, but in any case, Yerba Mate is also a major source of many important elements[8] for a healthy daily diet:

  > It contains vitamins A, C, E, B1, B2, Niacin (B3), B5, B... and complex minerals like Calcium, Manganese, Iron, Selenium, Potassium, Magnesium, Phosphorus, and Zinc. It also contains Carotene, Fatty Acids, Chlorophyll, Flavonols, Polyphenols, Inositol, Trace Minerals, Antioxidants, Tannins, Pantothenic Acid, and 15 Amino Acids.

- **Prolonged working hours:** Instead of having to leave the desk to get yet another cup of coffee, the knowledge worker can sit in front of his computer for hours, particularly when using thermos with a capacity of at least 1 or 1.5 liters (around half a gallon). Mate (the beverage) is also known for reducing appetite, which helps reduce costs in the case of companies providing food to their employees.
- **Teamwork benefits:** Given the inherent social origins of the habit of drinking mate, in the case of teams, or in the case of agile practices such as pair programming[9], sharing the mate (the recipient) helps team managers to create a sense of unity and common goal.
- **Increased sensitivity:** As with all caffeinated drinks, the intake of mate can lead to an improvement in the overall awareness[10] of the mate drinker.

Disadvantages

The following disadvantages of Mate (the herb, the beverage and the recipient) are worth considering:

- **Cold water effects:** Although common practice in Paraguay (where the infusion of Yerba Mate with cold water is known as Tereré[11]), this variant is known for causing violent reactions in the digestive system of the person drinking it, and it is strongly

[8] http://en.wikipedia.org/wiki/Mate_(beverage)#Health_Effects
[9] http://en.wikipedia.org/wiki/Pair_programming
[10] http://www.youtube.com/watch?v=Vi5JyCYKZws
[11] http://en.wikipedia.org/wiki/Tereré

recommended to never drink it more than 20 meters away from the nearest toilet.

- **Bitterness:** The strong taste of Yerba Mate is also a factor of considerable debate. Most mate drinkers usually start drinking it with sugar (some even with saccharine or other sweeteners), while most experienced drinkers will dismiss this practice and downplay those doing it as amateurish or otherwise ignorant. It is strongly recommended to have everyone agree on a mate variant beforehand to avoid shallow discussions on the relative merits of different approaches to mate drinking.
- **Mate lavado:** When the same Yerba has been poured several times (usually above 10 or 12 servings, depending on the quality of the Yerba), it loses part of its taste and must be replaced with new Yerba. Depending on how many people share the same mate (the recipient), this can be a significant problem, leading to reduced productivity and major anxiety and dismay.
- **"Matetiquette":** Mate (the beverage) is linked with a complete language, tied up to the history of the southern part of South America. As such, please be aware of the fact that serving a "mate lavado" (see previous item for an explanation of the concept) is considered rude practice, and is strongly discouraged. Serving mate with cold water, as explained above, can also be seen negatively, particularly if the person preparing the mate is not from Paraguay. Finally, talking in front of your recently-filled mate instead of drinking it, is also frowned upon, as you might be greeted with a "it's not a microphone" protest if you do it.

Conclusion

The importance of the Yerba Mate in the process of creation of software has been greatly dismissed by major research efforts, and we think that more research and mate drinking is needed. In our tests, Yerba Mate has been proven to foster creativity, teamwork, overall happiness, and trips to the toilets.

Roundup of iPhone App Sketchbooks

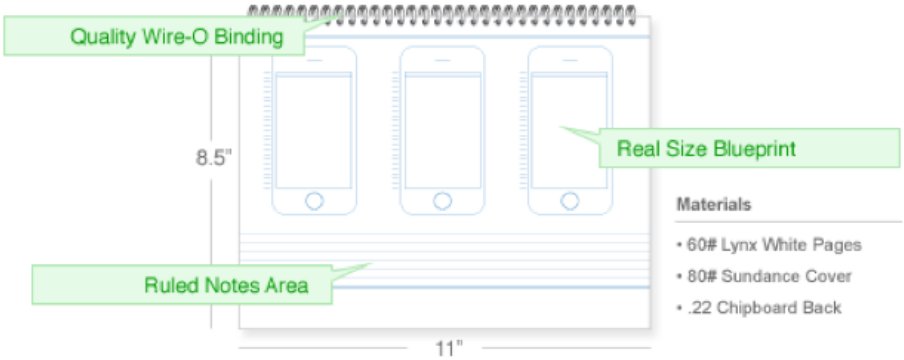
Adrian Kosmaczewski

2009-10-27

Nailing down the idea, the navigation and the UI of your next killer iPhone application is as important (if not more) as writing good code. This is why this post will showcase some recent iPhone designer products, all providing a paper-based, iPhone-shaped and iPhone-sized support for sketching out iPhone apps with your client (or just for your own creative pleasure).

Here we go:

App Sketchbook[1] by Square Position, LLC (USA)[2], @appsketchbook[3] on Twitter, sold via PayPal for USD 12.99, in both a perforated and non-perforated version.



iPhone Application Sketch Book[4] originally by Kapsoft (USA)[5] but recently sold to Apress Publishing[6], sold on Amazon for USD 9.99[7] (while supplies last!).

[1] http://appsketchbook.com/

[2] http://www.squareposition.com/

[3] http://twitter.com/appsketchbook

[4] http://www.mobilesketchbook.com/

[5] http://www.kapsoft.com/

[6] http://theappleblog.com/2009/09/29/iphone-application-sketch-book-rights-sold-to-apress-publishing/

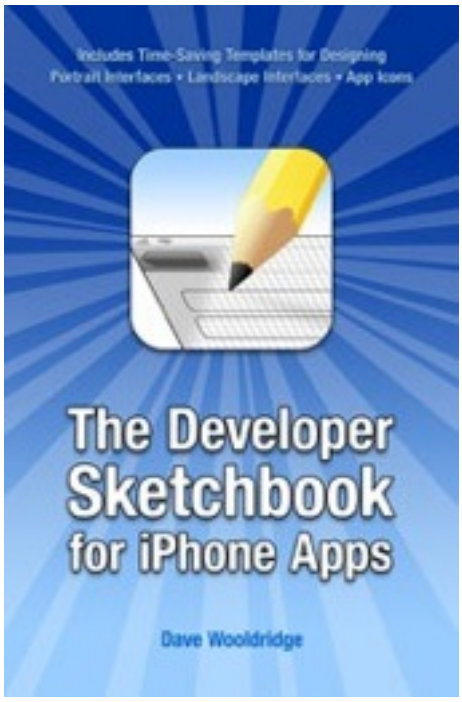
[7] http://www.amazon.com/iPhone-Application-Sketch-Book-Kaplan/dp/1430228237%3FSubscriptionId%3D0F0YTN83N46JSX6KDT02%26tag%3Dakosmasoftwar-20%26linkCode%3Dxm2%26camp%3D2025%26creative%3D165953%26creativeASIN%3D1430228237

Notepod[8] by Inventive Labs (Australia)[9], @notepods[10] on Twitter, sold via PayPal for USD 17.95.

The Developer Sketchbook for iPhone Apps[11] by Electric Butter-

[8] http://notepod.net/
[9] http://inventivelabs.com.au/
[10] http://twitter.com/notepods
[11] http://www.ebutterfly.com/books/devsketchbook/

fly (USA)[12], @ebutterfly[13] on Twitter,sold on Amazon for USD 19.99[14].



Finally, the **UI Stencils Sketch Pad**[15] (for USD 7.95) with its **iPhone Stencil Kit**[16] (for USD 17.95) by Design Commission[17] (@designcom[18] on Twitter).

[12] http://www.ebutterfly.com/

[13] http://twitter.com/ebutterfly

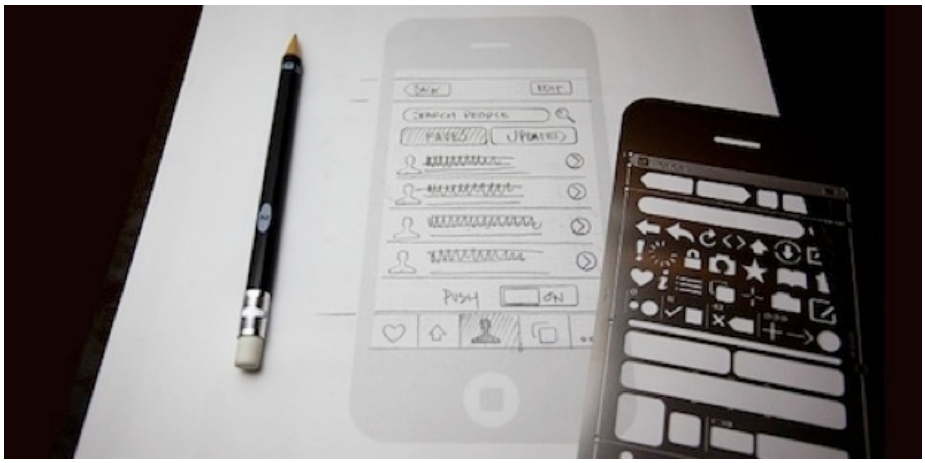
[14] http://www.amazon.com/Developer-Sketchbook-iPhone-Apps/dp/143925608X%3FSubscriptionId%3D0F0YTN83N46JSX6KDT02%26tag%3Dakosmasoftwar-20%26linkCode%3Dxm2%26camp%3D2025%26creative%3D165953%26creativeASIN%3D143925608X

[15] http://www.uistencils.com/featured/iphone-sketch-pad.html

[16] http://www.uistencils.com/featured/iphone-stencil-kit.html

[17] http://www.designcommission.com/

[18] http://twitter.com/designcom



I've ordered some of these items, and I've seen some others in the hands of some colleagues, and I can tell you that they are really handy and useful. I look forward to using them soon! For more information, here's a recent article on TUAW[19] comparing some of the items featured in this article.

Happy iPhone app designing!

Update, 2009-11-12: The iPhone Application Sketch Book[20] (recently bought by Apress) has been revamped and now features a plastic stencil too!

[19] http://www.tuaw.com/2009/10/03/mega-super-tuaw-shootout-of-the-iphone-ui-sketchbooks/

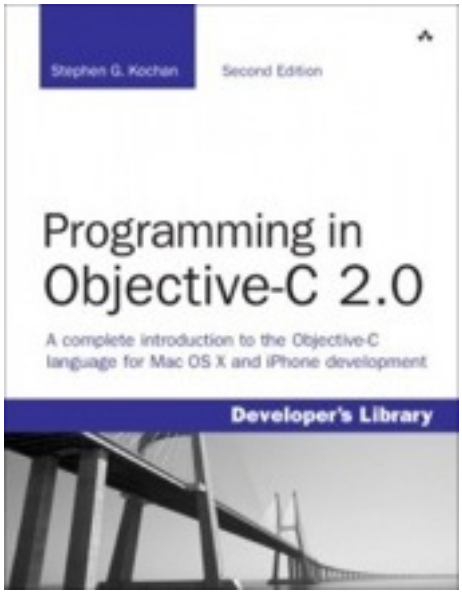
[20] http://www.mobilesketchbook.com/

iPhone Apps without Objective-C

Adrian Kosmaczewski

2009-10-29

Yes, it's possible. Even if Objective-C is one of my preferred programming languages[1], in any case I think it's worth mentioning that, 2 years after the official iPhone SDK has been announced[2], the iPhone development landscape has really grown up, and many, many different options are available today. This article provides a very high-level enumeration of some options I've found on the web, but I'm sure there are even more alternatives around.



Here it goes:

- First of all, remember that you can always create web apps[3]. It's worth mentioning that you can avoid the App Store and its quirks altogether; it's up to you ;) This opens up the possibility of using your preferred server-side technology + JavaScript, and there's quite a few libraries and tools that will help you create a killer

[1] /blog/preferred-programming-languages/

[2] http://www.tuaw.com/2007/10/17/apple-we-plan-to-have-an-iphone-sdk-in-developers-hands-in-fe/

[3] http://www.apple.com/webapps/whatarewebapps.html

web app: Joe Hewitt's excellent iUI[4] (yes, he's the same guy behind the Three20 project[5]), the Tank Engine Rails plugin Rails iUI plugin[6] (Tank Engine does not work very well unfortunately), iPhoney[7] or Eclipse[8] are just some of the alternatives.

- If you like C++, you can choose between the official SDK (yes, you can create iPhone apps with just C or C++), or other alternatives like nui[9], POCO[10], Boost[11] or haXe[12]. And apparently, soon you'll be able to use a Symbian C++ toolkit[13] as well (and who knows, maybe even we'll get Juce on the iPhone[14] one day too!).
- If you are a Flash and ActionScript developer, you are most probably aware that you can create native iPhone applications using Adobe Flash CS5[15]...
- For C# and .NET developers, there's MonoTouch[16], which has received extensive press coverage lately.
- If you want to use Lua, you can use the Wax framework[17].
- If your idea is to "write once, run anywhere", and "anywhere" in this context means Android, iPhone, BlackBerry and other mobile platforms, you might want to try rhomobile[18], Corona[19], PhoneGap[20] or QuickConnect[21]. The XMLVM[22] project might also interest you, as it consists of a cross-compiler toolchain which can be used with Ruby, .NET, Java and can generate code for many platforms at once.
- Finally, if you are into game development, the quantity of libraries allowing you to create iPhone games is simply astounding, many supporting alternative programming languages: SDL[23], Unity[24], SIO2[25], Torque2D[26], cocos2d[27] and Game

[4] http://code.google.com/p/iui/
[5] http://joehewitt.com/post/the-three20-project/
[6] http://github.com/noelrappin/rails-iui
[7] http://www.marketcircle.com/iphoney/
[8] http://www.ibm.com/developerworks/opensource/library/os-eclipse-iphone/
[9] http://libnui.net/
[10] http://pocoproject.org/blog/?p=208
[11] http://www.mani.de/backstage/?p=159
[12] http://drawlogic.com/2009/06/19/haxe-on-the-iphone-with-hxcpp-flash-9-api-to-c-for-mobile/
[13] http://www.daniweb.com/news/story228677.html#
[14] http://www.rawmaterialsoftware.com/juceforum/viewtopic.php?t=2832
[15] http://labs.adobe.com/technologies/flashcs5/appsfor_iphone/
[16] http://monotouch.net/
[17] http://github.com/probablycorey/wax
[18] http://rhomobile.com/
[19] http://www.anscamobile.com/corona/
[20] http://phonegap.com/
[21] http://quickconnect.sourceforge.net/browser/index.html
[22] http://www.xmlvm.org/iphone/#
[23] http://www.libsdl.org/index.php
[24] http://unity3d.com/
[25] http://sio2interactive.com/
[26] http://www.garagegames.com/products/torque-2d/iphone
[27] http://cocos2d.org/

Haxe[28].

- And of course, there's the official SDK[29], with Objective-C and Cocoa Touch all the big buzz around it.

What do you think? Is there any library or programming language that I've forgotten in this list? Just leave the name and URL in the comments below.

Update, 2009-11-05: Just found the Swebapps service[30] which allows you to create (very basic) iPhone apps without coding.

Update, 2009-11-24: Here's a new entry for the growing list of alternative frameworks: Crystal SDK[31] by Chillingo[32].

Update, 2009-11-25: Another one, in C++ and cross-platform: Airplay SDK[33].

[28] http://gamehaxe.com/category/nme/
[29] http://developer.apple.com/iphone/
[30] http://www.swebapps.com/
[31] http://www.crystalsdk.com/
[32] http://www.chillingo.com/
[33] http://www.airplaysdk.com/

HTTP Headers, Web Apps and Mobile Safari

Adrian Kosmaczewski

2009-10-30

I found today that Mobile Safari, the browser bundled with the iPhone, has a very strange and annoying behaviour when it comes to web apps. In fact, when you "install" web applications with the <meta name="apple-mobile-web-app-capable" content="yes" /> tag in the "Home Screen", **the USER_AGENT header sent to the server is different to the one sent when you access the same app manually using Safari.**.

Here's a test that proves this assertion: Create a server-side web app with your preferred language, and print all the request headers[1]; for example, in PHP:

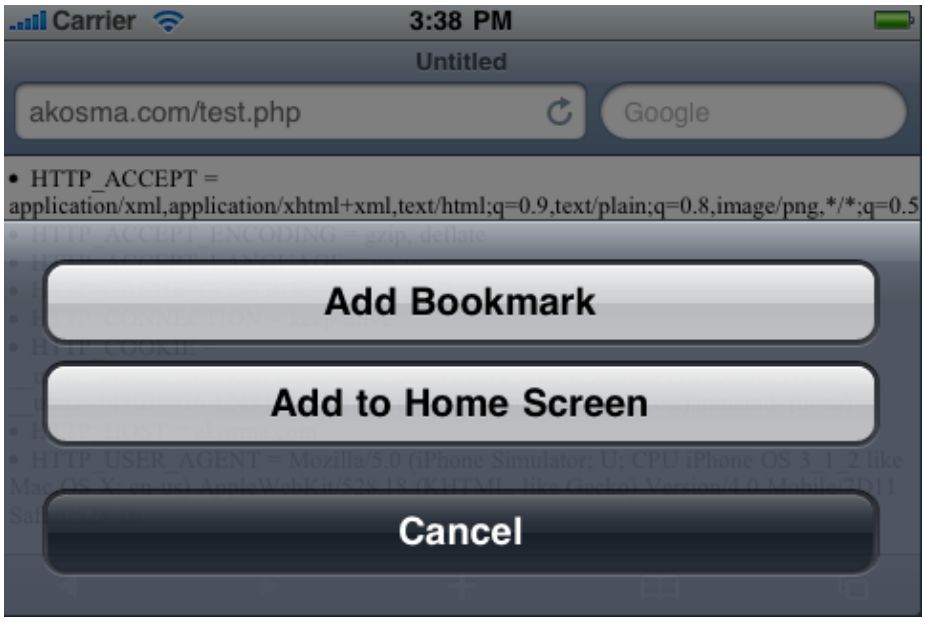
```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
        xml:lang="en" lang="en">
<head>
    <meta http-equiv="content-type"
            content="text/html;charset=UTF-8" />
    <meta name="apple-mobile-web-app-capable"
            content="yes" />
    <title>UserAgent</title>
</head>
<body>
<?php
foreach($_SERVER as $h=>$v) {
    if(ereg('HTTP_(.+)',$h,$hp)) {
        echo "<li>$h = $v</li>n";
    }
}
?>
</body>
</html>
```

Launch MobileSafari.app in either the iPhone Simulator or in your iPhone or iPod touch, and access the server-side script you created

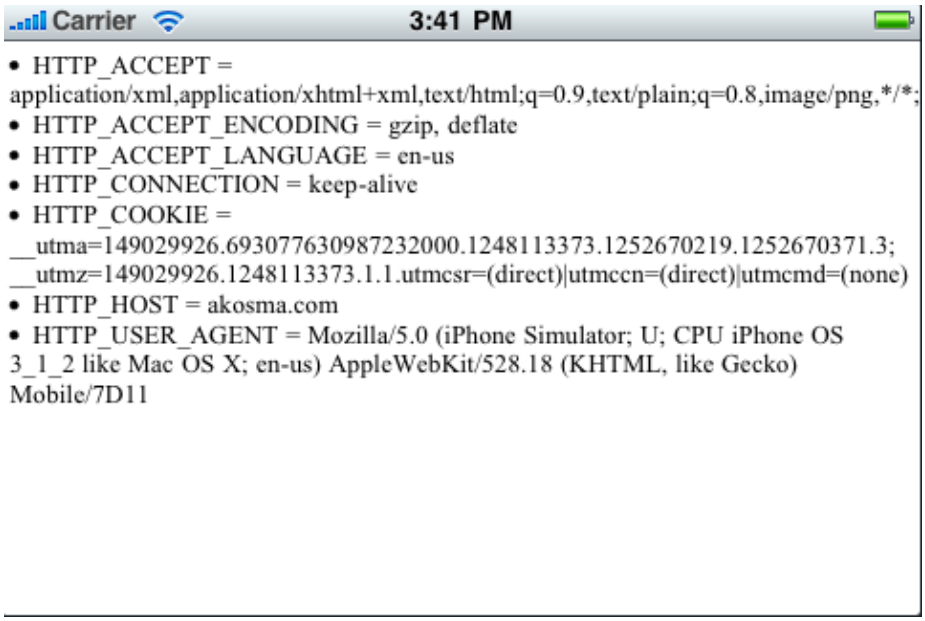
[1] http://tonycode.com/wiki/index.php?title=Dumping_HTTP_Headers

previously:

Add the application to your home screen, tapping the “+” button on the toolbar:



Now launch the application from your home screen, and this is what you see:



Compare the outputs (above, those from the latest iPhone Simulator, and below, those from my iPhone 3G with iPhone OS 3.1.2):

Safari:

```
HTTP_USER_AGENT = Mozilla/5.0 (iPhone; U; CPU iPhone OS 3_1_2 like
Mac OS X; en-us) AppleWebKit/528.18 (KHTML, like Gecko)
Version/4.0 Mobile/7D11 Safari/528.16
```

Home Screen App:

```
HTTP_USER_AGENT = Mozilla/5.0 (iPhone; U; CPU iPhone OS 3_1_2 like
Mac OS X; en-us) AppleWebKit/528.18 (KHTML, like Gecko) Mobile/7D11
```

This difference is only visible when the HTML generated by the server-side script contains the <meta name="apple-mobile-web-app-capable" content="yes" /> tag, which triggers this change of behaviour. Otherwise, the outputs are exactly the same. Most annoyingly, this behavior is not documented (at least not that I am aware of). This problem has already been spotted elsewhere[2] (scroll down until you see the comment by DVO published on Jul 3, 2009 at 7:34 AM; thanks to my friend Bertrand[3] for the link!).

This is not only weird, but it also might break libraries used to generate iPhone-ready websites out of existing web apps (by redirecting the browser using the information in the USER_AGENT header; for example, this is what happens with the current version of the Rails iUI plugin[4] (which is why I found this behaviour :)

[2] http://www.bennadel.com/blog/1197-Defaulting-To-The-Numeric-Keyboard-On-The-iPhone.htm

[3] http://twitter.com/bdufresne

[4] http://github.com/noelrappin/rails-iui/

The lesson of all of this mess is this: if you have to test for Safari on the iPhone, do not use the word “Safari”, but rather use “AppleWebKit” and “Mobile”, to be sure of catching also the users who installed your application in their home screen.

Update, 2009-10-30: In particular, the line of the Rails iUI plugin that causes problem with this particular behavior is this one[5].

[5] http://github.com/noelrappin/rails-iui/blob/50c7c19038817d30b4c351232217c87a065d18a1/lib/iphone_controller.rb#L45

Mention on StackOverflow (2)

Adrian Kosmaczewski

2009-11-04

Comment by JT[1] on question “technical way of managing files – grouping files in xcode”[2] in StackOverflow.com[3].

> Adrian Kosmaczewski explains an approach known to experienced Xcode developers. It’s the method I use and it “makes a place for everything and puts everything in its’ place”. Hope you find the article, linked below, what you’re looking for. Moreover, his blog is one of the best around.
>
> http://kosmaczewski.net/2009/07/28/code-organization-in-xcode-projects/[4]

[1] http://stackoverflow.com/users/209225/jt

[2] http://stackoverflow.com/questions/1718869/technical-way-of-managing-files-grouping-files-in-xcode

[3] http://stackoverflow.com/

[4] /blog/code-organization-in-xcode-projects

5 Years

Adrian Kosmaczewski

2009-11-06

Today it's the 5th anniversary of the article that would eventually become the first post of this blog[1]. I was leaving Buenos Aires, again, and I wrote that on my old G3 iBook in the airport of Ezeiza, right before boarding. That trip was very important, for many reasons that don't fit on a single blog entry.

Since then, I met Claudia, we got married, I finished my master degree[2] and started my own company[3]. Even hernún[4] came to Switzerland! I've moved from talking about .NET to giving interviews about the iPhone[5]. I've published as much text and code as I could, but most importantly, I kept on creating things.

Thanks to all of you, for your comments, your support, your ideas, your code, your critics. I've really learnt a lot during these years, and I hope my ramblings will be useful to you in the future as well.

Cheers! Salud! Santé!

[1] /blog/justo-antes-de-irme/
[2] /blog/master/
[3] http://akosma.com/
[4] http://hernun.com.ar/
[5] https://akos.ma/about/#_on_the_press

Thoughts about Google's Go Programming Language

Adrian Kosmaczewski

2009-11-12

Historically, we can distinguish really big software companies for providing, at least, four major kinds of products: an operating system (sometimes open sourced at a certain level), a web browser (with various degrees of standard compliance), a suite of office applications (slightly compatible with everyone else's), and a programming language with curly brackets (generally incompatible with everything else). In that particular order, we have:

- **Microsoft:** Windows, Internet Explorer, Microsoft Office, and C#.
- **Sun:** Solaris, HotJava (sic), StarOffice, and Java.
- **Apple:** Mac OS X, Safari, iWork, and Objective-C.
- **Google:** Chrome OS, Chrome, Google Docs, and... Go.

Precisely, Go[1] was the last piece that Google had to create in order to fit into the framework above. And it did, with a bright team including Ken Thompson (of Unix and C fame) and Rob Pike (of Plan 9 and UTF-8 fame). With names like that, and with Google's own funding and infrastructure, it is normal that the media went into a hype frenzy yesterday.

I think, however, that Google's engineers got tired of what the current and upcoming versions of their "official" programming languages (Java 7, C++0x and Python 3.0) had to offer, and simply came up with a programming language that fits better their needs and expectations. As one of the slides[2] of the TechTalk[3] says, with current languages "You can be productive or safe, not both."

Features like built-in support for concurrency or garbage collection hide the real true feature behind the language: faster build times with static typing support. This is important for Google from a software economy point of view: they want more productivity from their developers, or, in other words, more bang for their buck, all together with verifiable quality and speed of execution. Go seems to be designed to deliver in these areas. However, Rob Pike is careful to say

[1] http://golang.org/

[2] http://golang.org/doc/go_talk-20091030.pdf

[3] http://www.youtube.com/watch?v=rKnDgT73v8s

that the language is experimental, so time will tell if their efforts were worth it.

In any case, it is worth noting that there was a previous programming language called Go![4] (whose author even wrote a book about it[5]), and after an InformationWeek article[6] revealed this, a petition has started in the Go bug tracking[7], asking Google to change the name of the language, all in the name of Google's own "Don't be evil" motto.

[4] http://en.wikipedia.org/wiki/Go!_(programming_language)

[5] http://www.lulu.com/content/paperback-book/lets-go/641689

[6] http://www.informationweek.com/news/software/web_services/showArticle.jhtml?articleID=221601351

[7] http://code.google.com/p/go/issues/detail?id=9

akosma software has a new website

Adrian Kosmaczewski

2009-11-23

This is something I should have done much earlier, but hey, better late than never: akosma software has a new website and I'm happy to invite you to take a look at it.

Open Kosmaczewski will slowly become a more personal platform, as most of my future iPhone-related material will appear in the new akosma blog. After 5 years of operation, good old Open Kosmaczewski is by no means shutting down; but a new, exciting chapter starts here, definitely.

Thanks again to all of you for your amazing support, your comments and ideas. I look forward to continue serving you through my company, akosma software.

Welcome to the new akosma software website!

Adrian Kosmaczewski

2009-11-23

A new template[1] on our preferred blogging platform[2], a couple of tweaks here and there – one of the advantages of working on an open-source platform – and a thriving business landscape. Things are looking better than ever.

After only a few months of operation, akosma sofware is already moving towards its objective: to become a leading provider of cross-platform, multilingual software solutions, recognized on a global level.

Right now, akosma software is collaborating in some major projects, including the iPhone application for a major european bank, and another for an important watch brand. Coupled with other ongoing projects, like a book – more on this soon – and some upcoming presentations in conferences next year. Oh, and don't forget the iPhone development course for beginners[3] in Zürich on December 9th!

This space will feature more and more information. Stay tuned! The RSS feed[4] is yours to subscribe to. And as always, feel free to leave your comments below; we'd love to hear from you.

[1] http://basicmaths.subtraction.com/
[2] http://wordpress.org/
[3] http://www.facebook.com/event.php?eid=195431175488&index=1
[4] http://feeds.feedburner.com/akosmasoftware

That Mobile Programming Mess

Adrian Kosmaczewski

2009-11-25

Let's be honest. 3 months ago I said that it wasn't a good time to be an iPhone developer[1]. Today, it looks like being one sucks every day a bit more. This article will dive into the alternatives, horrors, breakpoints, misconceptions, IDEs, sadness, hope, USB cables, and all those different factors that are shaping this thing called mobile software development market.

1) iPhone

First we had Rogue Amoeba[2] and Joe Hewitt[3] leaving the App Store, and then we have the autobot in the review process[4] rejecting apps including Joe's Three20 library[5] (which is a highly ironic fact, if you think about it), and then rejecting apps with method or property names apparently similar[6] to those of private APIs, then app rankings going bezerk[7] for a little while, then the release dates having hiccups[8], and so on, and so on.

All of this sucks. Ask Paul Graham[9] if you don't believe me. And check the improvements suggested by Enormego[10]. There is a consensus about the fact that things aren't working – at least from the developers' point of view.

---

[1] /blog/risk-management-in-iphone-projects/

[2] http://www.rogueamoeba.com/utm/2009/11/13/airfoil-speakers-touch-1-0-1-finally-ships/

[3] http://www.techcrunch.com/2009/11/11/joe-hewitt-developer-of-facebooks-massively-popular-iphone-app-quits-the-project/

[4] http://gizmodo.com/5405978/iphone-apps-have-to-be-approved-by-robots-now-too

[5] http://groups.google.com/group/three20/browse_thread/thread/c442af6e39a918b0/6d5046771539d139

[6] http://twitter.com/zenoc/status/5856233410

[7] http://www.tuaw.com/2009/11/04/developers-report-a-moment-of-upside-down-app-rankings-now-retu/

[8] http://www.theiphoneblog.com/2009/11/24/itunes-app-store-release-date-sorting-sorta-broken/?utm_source=feedburner&utm_medium=feed&utm_campaign=Feed%3A+TheIphoneBlog+%28The+iPhone+Blog%29&utm_content=Google+Reader

[9] http://paulgraham.com/apple.html

[10] http://developers.enormego.com/view/appstore_improvements?utm_source=feedburner&utm_medium=feed&utm_campaign=Feed%3A+EnormegoDevelopers+%28Enormego+Developer+Blog%29&utm_content=Google+Reader

However, let's be even more honest, things don't look better on the other side of the fence.

2) Web Apps

According to Peter-Paul Koch[11] (of Quirksmode[12] fame, all in all a highly respected web developer), the problem of the App Store is not Apple, but the stupid developers who insist in writing native apps when they could just write web apps using good ol'HTML 5 and CSS.

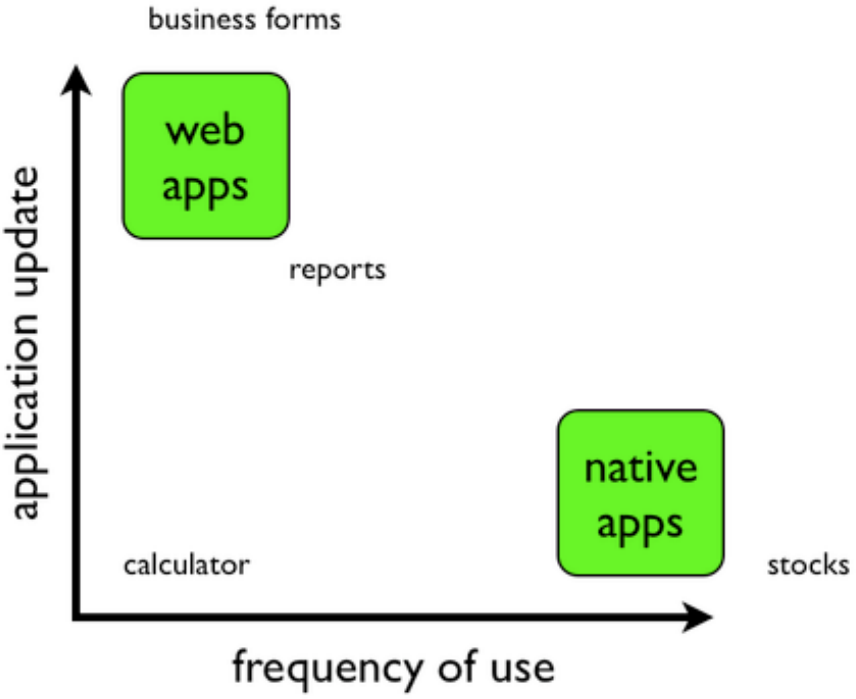
I agree that for a small range of applications, this approach would work, going from simple to-do lists, and even some social networking applications[13]. I've enumerated the option of web apps in a recent article of mine[14] about alternatives to the native SDK.

But, there's always a "but":

- The money is in native iPhone applications. My clients, at least, they want them, they are ready to pay for them, just like they paid for websites back in 1997. And the monetization of iPhone apps is simply brilliant. Submit your app, cross your fingers, charge a few bucks, and you have potentially 50 million clients ready to buy it. Try finding a monetization model for your web app based entirely on the iPhone.
- Games: although I'm not a game developer (yet) at least for the moment it's plain unthinkable to create fun, appealing games using HTML and JavaScript.
- Hardware access: you can access the GPS information, but that's more or less everything you can do. Try to access the camera, the local network stack (think Bonjour-based apps), or the microphone, or the speakers, or the iPod music library, or the address book of the user, or the accelerometer, or complex UI animations or transitions, and you're toast. The point is, many complex apps are based on those elements. Complex user experiences involve some or all of these elements, not just plain data. I do hope that Apple will allow access to some of these native features in the iPhone soon, particularly the accelerometer and the camera, which would be in my opinion perfect candidates for a JavaScript API.

There certainly is a business case for web apps, as I explained in my presentation last year at Geneva's iPhone Conference 2008:[15]

[11] http://www.quirksmode.org/blog/archives/2009/11/apple_is_not_ev.html
[12] http://www.quirksmode.org/
[13] http://googlemobile.blogspot.com/2009/07/google-latitude-now-for-iphone.html
[14] /blog/iphone-apps-without-objective-c/
[15] /blog/iphone-conference-2008/



In short: yes, web apps are a perfect alternative in some cases; yes, Apple is unpredictable and the App Store must be improved; and finally no, developers aren't stupid. There's a great deal of personal taste in the choice of a development platform, and in this case, the tradeoff is sometimes worth the pain.

John Gruber says it even better than me[16]:

> But the best proof is what I pointed out above: Apple itself created almost no iPhone web apps. Successful iPhone developers don't just want to write software that works on the iPhone. They want to write software for the iPhone that's just as good as Apple's. Today that means using Cocoa Touch and the native SDK.
>
> When you write a Cocoa Touch app for the iPhone, you're not starting from scratch. You're starting with the Cocoa Touch framework. As Faruk Ateş astutely points out in his response to Koch, to discount the framework is to discount everything that sets the iPhone apart as a development platform. Not only are native iPhone apps faster and more capable than their web-app equivalents, but they're easier to write.

Anyway.

[16] http://daringfireball.net/2009/11/iphone_web_apps_alternative

3) Android

I think, without any doubt, that Android is one of the most important platforms of the past few years (I agree with Tim Bray[17] in this point). Android has many characteristics that are, at first sight, extremely appealing:

- It's open source, based on open standards and proven technologies;
- It's backed by a company known by its technical expertise and innovation;
- It's growing faster and faster every day;
- It is proving popular among users, too – and this is not a minor element!

But, of course, not everything is green: Android basically trades off a shitty review process with a shitty programming environment (at least until IntelliJ IDEA 9[18] ships*). Not only that: the whole platform is threatened by Google releasing its own device[19], or by the fragmentation of the platform[20] by the device vendors, or by the low numbers of app sales[21] (which is somewhat surprising, given the apparent strong sales of some devices like the Droid[22]).

Of all the drawbacks I enumerated above, I want to focus in one aspect: the problem of platform fragmentation is, in my opinion, so big, so important, that voices are raising to explain[23] that this single factor might blow the whole platform:

> Can you write an .apk application that runs on all devices? Theoretically, yes. But not without testing on an ever-increasing number of gadgets. This is the problem that Symbian and J2ME phones have, and the road that Android is headed down if Google doesn't reign in control and quickly. Differing OS versions, different manufacturer and carrier customizations, and various app stores are going to hobble the OS before too long.
>
> All that said, I still believe it's got a real future after using it regularly on my Archos, but Google needs to get control quickly. I had originally suggested using the Android logo and trademark (which they may or may not own) as a way of ensuring compatibility, but it seems the logo is creative

[17] http://www.tbray.org/ongoing/When/200x/2009/11/20/Android-Splintering

[18] http://www.jetbrains.com/idea/nextversion/#Android_Development

[19] http://www.techcrunch.com/2009/11/17/thegoogle-phone/

[20] http://www.theiphoneblog.com/2009/11/17/fake-steve-android-fragmentation-harder-develop-iphone/

[21] http://us.mobile.reuters.com/m/FullArticle/p.rdt/CTECH/ntechnologyNews_uUSTRE5AJ1EU20091120

[22] http://www.pcmag.com/article2/0,2817,2355916,00.asp

[23] http://www.russellbeattie.com/blog/android-is-splintering-just-not-how-you-think-it-is

> commons. So maybe they need to come up with an "Android Approved" logo or something.

At least in terms of programmer experience, though, Android still has a long way to go. To begin with, it's based in Java; and I will never get tired of quoting this paragraph from His Highness Steve Jegge[24] (Google employee, by the way):

> I'll give you the capsule synopsis, the one-sentence summary of the learnings I had from the Bad Thing that happened to me while writing my game in Java: if you begin with the assumption that you need to shrink your code base, you will eventually be forced to conclude that you cannot continue to use Java. Conversely, if you begin with the assumption that you must use Java, then you will eventually be forced to conclude that you will have millions of lines of code.

You can't be more clear. More code leads to higher maintenance costs, higher chances of bugs, and in general, harder-to-maintain code bases. The choice of Java[25] in the long run will hurt Android more than anything else. In a similar way, the current App Store policies are doing a similar harm, which is, as Graham said, to make developers run away from either platform.

Additionally, I might also point out the fact that currently the only way to write Android apps, is using Java (with a small subset of functionality available through the Android Native Development Toolkit[26]); compare this with the amount of alternative frameworks for the iPhone[27], and the discussion of the "openness" of the Android platform takes a different spin.

Finally, there's this factor Gruber discusses[28], this syndrom called the "year of Android", similar to the "year of Linux" promised since the early years of this decade... and which every year turns out, inexorably, in favor of other platforms.

> I'll just say that if the consensus winds up that the Droid isn't a great Android phone, this is the sort of attitude that'll sink Android. It's the same attitude desktop Linux has always had, that the future is going to be great, so don't worry about the present.
>
> Like a sports team that's always saying "Wait until next year", meanwhile, Apple has won another championship this year.

[24] http://steve-yegge.blogspot.com/2007/12/codes-worst-enemy.html
[25] http://twitter.com/stevedekorte/status/5902464622
[26] http://developer.android.com/sdk/ndk/1.6_r1/index.html
[27] /blog/iphone-apps-without-objective-c/
[28] http://daringfireball.net/linked/2009/11/22/bray-android

4) The Others

Just to name them, a bit further away in the finish line, we have a Windows Mobile[29] base of millions of devices without any serious roadmap, and which looks more like a neglected child than anything else. Or the Palm Pre which is, hum, nonexistent (at least in this side of the world). Or BlackBerry, which is, hum, off the radar (but which I think is the real third alternative to the iPhone and Android). Or Symbian and J2ME, which are, well, Symbian and J2ME.

So, in short, this means that programming for mobile platforms is both shitty and complex, no matter how you look at it.

Great.

Meanwhile, in a small town in Switzerland…

akosma software does and will concentrate mobile development efforts on the iPhone and Android platforms. We think, given all the facts we could chew so far, that these two are more likely to dominate the mobile software development business in the years to come than all the others combined.

Nevertheless, in terms of absolute and relative numbers, current projects and cash at stake, iPhone is the clear winner at the time of this writing (something said a couple of months ago in an interview on the Swiss newspaper Le Temps[30]).

In terms of potential for growth, we cannot and we do not underestimate Android (hence our interest). Particularly, there are some factors that could boost Android's appeal, which are huge opportunities for growth, somewhat neglected so far:

- Google should promote the platform more, including ads on TV, and also allowing paid apps in more countries;
- Developers should be capable of creating Android apps in as many languages as possible. Compiled or not, scripting or not, object-oriented or not, it doesn't matter – I vote for C++ and Ruby;
- We need a better IDE, now (and a faster emulator, too. The current one sucks).

Above all, there's one important fact: programming is still fun, creative and interesting. And we'll keep on creating new stuff (and ranting about it), no matter what platform we choose.

(*) I have tried the current beta, and I don't think[31] it will change the game that much. No visual GUI editor (something that even

[29] http://reddevnews.com/blogs/desmond-file/2009/01/microsofts-mobile-mess.aspx
[30] /blog/interview-sur-le-temps/
[31] http://twitter.com/akosma/status/5949948423

Eclipse has, albeit in a primitive way), and the configuration of a single project, well, it's IntelliJ IDEA[32]. I couldn't even run a simple project with it. Let's see when the final release ships.

Update, 2009-11-26: This article in CNN[33] (French version)[34] talks about the problem platform fragmentation represents for small shops. It brings a strange feeling: the "open" Android platform might be only open to those with enough means to have a whole testing framework with the most popular devices, leaving out small shops who wouldn't be able to support their applications in all devices, given the famous fragmentation.

[32] /blog/not-exactly-what-i-meant/

[33] http://edition.cnn.com/2009/TECH/11/17/android.wired/index.html

[34] http://www.pcinpact.com/actu/news/54322-android-croissance-inquietudes-developpeurs-smartphones.htm

iPhone Training in Zürich

Adrian Kosmaczewski

2009-11-26

Trifork AG[1] and akosma software[2] are organizing a one-day, 5-hour training in Zürich about iPhone development! **The iPhone Course for Beginners** takes you on a 5 hour trip, from creating your first Xcode project to submitting an application to the App Store.

This training is targeted to web, J2EE and .NET developers looking to get a hands-on introduction to the new world of Cocoa Touch, the iPhone, Xcode and all of this whole Apple frenzy. It can also help future iPhone project managers to understand the complexity and requirements of their future projects.

It will take place Wednesday, December 9, 2009 from 4 PM to 9 PM in Zürich (the exact location will be announced later).

Contents:

- Introduction to Cocoa Touch and Objective-C;
- Creation of an Xcode project;
- iPhone application design: design patterns, best practices, tips and tricks;
- Use of Interface Builder to create user interfaces;
- Application architecture: the Model-View-Controller paradigm;
- iPhone features:
  - Using touch events;
  - Using the accelerometer;
  - Accessing the camera;
  - Getting location information from the GPS;
  - Sending e-mail from an application;
- Preparing an app for deployment:
- Ad-Hoc deployment procedures;
- Testing the application on the device;
- Adding an icon and a default image;
- Submitting an application to the App Store;
- Maintenance tasks:
  - Refactoring;
  - Documenting;
  - Adding unit tests;

[1] http://www.trifork.com/
[2] http://akosma.com/

The course includes a USB key with a PDF booklet including the whole contents of the course, plus the complete source code of the application created during the session.

Prerequisites:

- Previous programming knowledge in any object-oriented language;
- Basic knowledge of the Mac OS X user interface.

Requirements:

- A MacBook computer;
- Mac OS X 10.6 “Snow Leopard”;
- Xcode 3.2;
- iPhone SDK 3.0 or higher

If you have an iPhone Developer Program account, feel free to bring your iPhone and USB cable to the course too, but this is not mandatory.

Price: CHF 150.00 for attendees of the iPhone Developer Days[3] in October 2009 in Zürich, and CHF 200.00 for those who haven’t attended the seminar.

To register, choose one of the following options:

- Send an email to sec@trifork.com[4];
- Mark your RSVP status in the Facebook Event[5];
- Or in the LinkedIn Event[6].

[3] http://iphonedevday.com/suisse-2009/
[4] mailto:sec@trifork.com
[5] http://www.facebook.com/event.php?eid=195431175488
[6] http://events.linkedin.com/Building-first-iPhone-application/pub/172468

WordPress

Adrian Kosmaczewski

2009-11-27

I am a big WordPress fan. And I do not suffer from the NIH[1] syndrome. So when I had to build a site for my company, the choice of WordPress was a natural one. I've been using it for Open Kosmaczewski[2] (my personal blog now) for over 3 years, and I must say that it impresses every day a bit more.

For those wondering what's behind the hood in this new WordPress blog, this blog / site displays the Basic Maths[3] theme for WordPress, designed by Khoi Vinh[4] & Allan Cole[5] (to all RSS subscribers out there: I recommend taking a look at their theme. It's simply beautiful).

As for the plugins, here's the full list, and I use a lot of them, indeed:

- Akismet[6] (duh)
- All in One SEO Pack[7]
- Amazon Reloaded for WordPress[8]

[1] http://en.wikipedia.org/wiki/Not_Invented_Here
[2] http://kosmaczewski.net/
[3] http://basicmaths.subtraction.com/
[4] http://subtraction.com/
[5] http://fthrwght.com/
[6] http://akismet.com/
[7] http://semperfiwebdesign.com/
[8] http://plugin-developer.com/amazon-reloaded-for-wordpress/

- Contact Form 7[9]
- Email Log[10]
- FeedBurner FeedSmith[11]
- Google Analytics for WordPress[12]
- Google XML Sitemaps[13]
- Lightbox 2[14]
- List Subpages[15]
- Page Links To[16]
- Really Simple CAPTCHA[17]
- Similar Posts[18]
- Simple Google Map[19]
- Simple Sidebar Share Widget[20] (which I customized a bit)
- Subscribe To Comments[21]
- SyntaxHighlighter Evolved[22] (which I had to tweak a bit, given that the support for Objective-C is currently broken off the box, which as you can imagine is a bummer for an iPhone dev blog like this one :)
- Technorati Incoming Links[23]
- Twitter Tools[24] and its accompanying plugins: Bit.ly URLs[25], Exclude Category[26] and Hashtags[27]
- WordPress.com Stats[28]
- WP-DBManager[29]
- WP-Mail-SMTP[30]
- WP Super Cache[31]
- WPtouch iPhone Theme[32], which powers the mobile version of

[9] http://contactform7.com/
[10] http://sudarmuthu.com/wordpress/email-log
[11] http://www.feedburner.com/fb/a/help/wordpress_quickstart
[12] http://yoast.com/wordpress/analytics/#utm_source=wordpress&utm_medium=plugin&utm_campaign=google-analytics-for-wordpress
[13] http://www.arnebrachhold.de/redir/sitemap-home/
[14] http://www.stimuli.ca/lightbox/
[15] http://robm.me.uk/projects/plugins/wordpress/list-subpages/
[16] http://txfx.net/code/wordpress/page-links-to/
[17] http://ideasilo.wordpress.com/2009/03/14/really-simple-captcha/
[18] http://rmarsh.com/plugins/similar-posts/
[19] http://clarknikdelpowell.com/wordpress/simple-google-map/
[20] http://wordpress.org/extend/plugins/simple-sidebar-share-widget/
[21] http://txfx.net/code/wordpress/subscribe-to-comments/
[22] http://www.viper007bond.com/wordpress-plugins/syntaxhighlighter/
[23] http://www.mydigitallife.info/2007/10/10/technorati-incoming-links-plugin-for-wordpress/
[24] http://alexking.org/projects/wordpress
[25] http://crowdfavorite.com/wordpress/
[26] http://crowdfavorite.com/wordpress/
[27] http://crowdfavorite.com/wordpress/
[28] http://wordpress.org/extend/plugins/stats/
[29] http://lesterchan.net/portfolio/programming/php/
[30] http://www.callum-macdonald.com/code/wp-mail-smtp/
[31] http://ocaoimh.ie/wp-super-cache/
[32] http://bravenewcode.com/wptouch

this site.

Kudos to all the WordPress plugin developer community! There's really an astounding amount of high quality plugins out there. Amazing stuff.

Finally, and somewhat related and unrelated with WordPress at the same time, I've also become a keen user of MarsEdit[33], which I am using to write articles for my both blogs. I don't know why I haven't bought a copy of this brilliant software before. If you're any serious about blogging, get your copy of MarsEdit now (disclaimer: I'm not affiliated with Red Sweater Software LLC[34] in any way, I'm just a very, very happy customer).

[33] http://www.red-sweater.com/marsedit/
[34] http://www.red-sweater.com/

Premier Chapitre Android Application

Adrian Kosmaczewski

2009-11-30

akosma software is proud to announce the availability of PremierChapitre for Android in the Android Market[1]. PremierChapitre is the first Android application developed in partnership with Stephan Burlot of Coriolis Technologies[2].

This application is a port of the iPhone application[3] (developed by Stephan) of the French service Premier Chapitre[4]. This service, as the name implies, allows users to read the first chapter of newly released books.

[1] http://www.android.com/market/
[2] http://www.coriolis.ch/
[3] http://url.akosma.com/rv2zqj
[4] http://www.premierchapitre.fr/

Given the lack of an official web repository of Android apps providing URLs and previews, you can find below some links to popular Android sites featuring the application description and some screenshots:

- AndroLib.com[5]
- AndroidZoom[6]
- Android Software[7]

[5] http://www.androlib.com/android.application.fr-premierchapitre-androidapp-znix.aspx

[6] http://www.androidzoom.com/android_applications/lifestyle/premierchapitre_bcsr.html

[7] http://www.android-software.fr/premierchapitre

Two conferences and one training

Adrian Kosmaczewski

2009-12-06

This week I will spend a good deal of time in one of my preferred cities: Zürich. On Tuesday I will have the privilege of talking at this week's Webtuesday meeting[1] at Liip's offices[2]. Then, on Wednesday, I'll be giving the 5-hour iPhone training I've blogged before[3], and which has been so far an unprecedented success! Stay tuned for future editions of this training in 2010.

At the Webtuesday meeting I'll be talking about web frameworks for iPhone and mobile development, some of which I've referenced on my previous post[4] about alternative iPhone development frameworks. I will publish the slides, as usual, on SlideShare[5] after the meeting. Be sure to add your name to the wiki page[6] if you are coming! I hope to meet you there.

But tomorrow Monday I'll begin a great week attending TEDx Geneva[7]! I look forward to watching great conferences and also meeting in person many of you.

[1] http://webtuesday.ch/meetings/20091208
[2] http://www.liip.ch/
[3] /blog/iphone-training-zurich/
[4] /blog/iphone-apps-without-objective-c/
[5] http://www.slideshare.net/akosma/presentations
[6] http://webtuesday.ch/meetings/20091208
[7] http://www.tedxgeneva.com/

Who Do You Want to Work With?

Adrian Kosmaczewski

2009-12-07

When you are a kid in Argentina, there are invariably three questions that you'll always get asked whenever you meet a grown up person:

- How old are you?
- What's your favorite football team?
- What do you want to be when you grow up?

The answer to the first question depends on the moment, of course, and it's simply a test to see if you know how to count. The answer to the second depends on your parents (this is like religion down there) and the city where you live (but there's a 90% chance your answer will be either River Plate[1] or Boca Juniors[2]).

The third question, however, is problematic, no matter what the answer is. Because at a large degree we build our lives around that "what do you want to be?" question, whether we like or not what we do, whether we believe or not that what we want to do is doable or not, or if it pays well or not, or if we will like at all, or if we will end up doing something completely different whatsoever by the time we retire.

This single question shapes a lot our lives, without even realizing it, and we pollute otherwise peaceful kids with the realization that there's much more to life than school and Wii and friends and chocolate milk.

The problem is, for me this is clearly the wrong question to ask. **We should be asking kids "who do you want to work with?", instead.**

There's an old adage that goes like this: "It's not what you know, it's who you know". Life is made of relationships, not pure knowledge got through 12 boring years of study, in a terrible environment made of vertical authorities and obligatory dissertations about horribly dull subjects.

In this blog I've often held the hypothesis that most problems in software teams are not technical, that the technical problems have been

[1] http://en.wikipedia.org/wiki/Club_Atlético_River_Plate
[2] http://en.wikipedia.org/wiki/Boca_Juniors

solved long ago. I think that the root cause for many software problems are social problems, like team cohesion or communication. Likewise, in terms of social relationships, in terms of society, the root causes for our problems lie in our capacity to understand each other.

And in spite of all the efforts and money spent every year in workplace health problems, in spite of all the deaths[3] and the acknowledgement of the existence of assholes[4] in our jobs, we still ask our kids "what do you want to be when you grow up".

This question seems harmless by itself, but looking closely, it supports several fallacies:

- It implies that all professions are carried out similarly whatever you choose to be and whenever you work; that is, you will be as happy and able to carry out your craft whether you are an independent SAP consultant, a blacksmith in Patagonia or if you are a freshly graduated doctor on internship at Seattle's Grace Hospital[5]. This is simply not true. The amount of initiative, self-drive, learning opportunities, job stability, are simply not the same, and lots of people learn this really late. I know it too well: I have learnt it quite late.
- It implies that you will not be independent. When you ask the kid about professions, you are not expecting an answer like the previous "blacksmith in Patagonia" thing, because in our society, being able to ask such a question means that you have the means to send your kids to school (or at least you expect to do so). If you live in a "favela" in Rio de Janeiro, and you barely have enough to eat, you don't think that much about the future. And if you do, the kid will most probably answer that he wants to be a football player or a TV star. In these places and situations (where most of mankind actually lives and dies every day) the present moment eats all your CPU time. Which means that the medium-class kid will most probably answer with the name of a profession, and apart from some honorable exceptions, universities don't teach how to be independent, but just how to be another animal in the herd. Which cuts off lots of possibilities, needless to say.
- It implies that social groups built around different professions are all comparable. They are not. They do not have the same motivations or ethics. I've studied physics (a rather nerdy career) and then switched to economics (a supposedly rather trendy one). The groups of people that gather around professions are not at all comparable, and I must say that I've finally settled down working in a field where, as Jeff LaMarche[6] told me once, "the asshole ra-

[3] http://www.guardian.co.uk/world/2009/sep/09/france-telecom-staff-suicides-phone

[4] http://bobsutton.typepad.com/

[5] http://en.wikipedia.org/wiki/Grey\XeTeXglyph\numexpr\XeTeXcharglyph"0027\relax{}s_Anatomy

[6] http://iphonedevelopment.blogspot.com/

tio is astonishingly low". You couldn't say it better. And I couldn't be happier about it.

- Finally, it implies that the kids know how much money you get in different jobs. This is not a minor point; if a kid wants to choose a path depending on economic reasons, she should be able to do so without external input. I know too many people who regret not choosing a path over another because they did not know the market conditions of each one.

In short, we do not teach our kids to question the world we're living in, to search for lots of answers before taking decisions, and also, to question the authority, simply because we haven't been taught to do that. And given the natural human inertia to avoid change, it is somehow natural that our questions to kids reflect what our parents asked us. We tend to repeat mistakes, and that not only works at a macro level, but also at micro level. It is part of human nature.

By asking "who do you want to work with" you ask explicitly your kid to choose between allies and assholes, and given that choice, guess who most kids will choose? The question will also prompt them to learn to accept their own choices, too, because it prompts a thought process much more elaborated than just answering "doctor" or "lawyer" or any other similar politically correct answer.

Kids are much more intelligent than we are, and then we send them to school to avoid having them telling us repeatedly how stupid we are.

Finally, by asking "who do you want to work with?" you are also implicitly asking "who you don't want to work with?", which is the second most important question you should ask, and whose answer is not implied by the first. Dilbert is a funny comic, but I don't see why we keep on behaving that way, when our working life could be much more enjoyable, by any standards.

Try it: if you have a kid, or the next time you meet one, ask her or him this very two questions. You will be surprised of the answers, and you might as well learn something about your own life, too.

Webtuesday: Mobile App Frameworks

Adrian Kosmaczewski

2009-12-09

I had great fun yesterday at Liip[1]'s offices, during the monthly Webtuesday meeting[2]. The presentation, which consisted in a high-level overview of mobile web application frameworks and platforms, gathered around 20 people, and the conversations and discussions (and the beers) after the talk were great. I hope that everyone enjoyed the evening as much as I did!

These are the slides I used during my presentation. They are released with a Creative Commons Attribution-Noncommercial-Share Alike 2.5 Generic[3] license, so feel free to share them, always with a link back to the original SlideShare URL.

[1] http://www.liip.ch/

[2] http://webtuesday.ch/meetings/20091208

[3] http://creativecommons.org/licenses/by-nc-sa/2.5/

Siegfried Ceballos: The Idea Detective

Adrian Kosmaczewski

2009-12-10

Siegfried Ceballos[1], the "idea detective" (@ideadetective[2] on Twitter), is publishing an interesting series of short videos, asking people about the estimated percentage of ideas they are able to actually bring to reality. The answers to this very atypical and interesting question are insightful and different, and I had the pleasure to answer[3] it to Siegfried last Monday at a break during the TEDx Geneva[4] conference.

> Adrian Kosmaczewski, Akosma Software, Pully, answered to the question in French: "What percentage of ideas are actually carried out?". He says 15% is a reasonable figure. He does not have the time to carry out more but he writes his ideas down on his paper notepad.
>
> The interview was made at TEDxGeneva, Monday 7 December 2009.

The question is interesting in many respects, because, as I said in the video, many of my ideas are simply not doable in a reasonable timeframe, and this leads to the appearance of "derived", "doable" ideas, and some of these are about software, while other ideas are more about just doing business in other fields. In any case, my Moleskine[5] notepad is my friend.

In the same series, don't miss the answers from Joan Flanagan[6] from Saatchi & Saatchi, Michael Doser[7] from CERN, Bruno Giussani[8] from TED, Christophe Gras[9] from Hortis, Luciano Benassi[10] from SoSoft-

[1] http://www.csid.ch/

[2] http://twitter.com/ideadetective

[3] http://www.csid.ch/site/2009/12/09/ideas-done-adrian-kosmaczewski-akosma-software-pully/

[4] http://www.tedxgeneva.com/

[5] http://www.moleskine.com/

[6] http://www.csid.ch/site/2009/12/09/ideas-done-joan-flanagan-saatchi-saatchi-geneva/

[7] http://www.csid.ch/site/2009/12/09/ideas-done-michael-doser-cern-geneva/

[8] http://www.csid.ch/site/2009/12/08/ideas-done-bruno-giussani-ted-conferences-geneva/

[9] http://www.csid.ch/site/2009/12/04/ideas-done-christophe-gras-hortis-grc-petit-lancy/

[10] http://www.csid.ch/site/2009/12/04/ideas-done-luciano-benassi-sosoftware-lausanne/

ware and Frederic Sidler[11] from Mixin.com.

Thought-provoking! Thanks to Siegfried for this.

[11] http://www.csid.ch/site/2009/12/02/ideas-done-frederic-sidler-mixin-com-lausanne/

Upcoming Conferences: SDC in Göteborg and iPhone Dev Days in London

Adrian Kosmaczewski

2009-12-11

2010 will be for akosma software, without any doubt, the year of the conference.

I am delighted and absolutely thrilled to announce that there already are two major events in the agenda for next year, where I will have the privilege to share the stage with major names in the field, to talk about this passion, namely, iPhone software development:

[1]

First, there's the Scandinavian Developer Conference 2010 in Göteborg[2], Sweden, on March 16th & 17th, 2010. This event features this year a mobile technologies track[3], and I will be giving an introduction to iPhone development, as well as a more in-depth discussion about consumption of REST services from iPhones. In the same conference there will be sessions about MonoTouch[4] and Android[5], so all in all it is an event I'm really looking forward to attend.

[6]

Then, the following month, the iPhone Dev Days 2010 in London[7], or-

[1] http://www.scandevconf.se/
[2] http://www.scandevconf.se/
[3] http://www.scandevconf.se/2010/conference/tracks/#mobile-solutions-1
[4] http://scandevconf.se/2010/conference/speakers/paul-rayner/
[5] http://scandevconf.se/2010/conference/speakers/marko-gargenta-marakana/
[6] http://iphonedevday.com/iphone-london-2010/
[7] http://iphonedevday.com/iphone-london-2010/

ganized by Trifork[8] and sponsored by O'Reilly[9], taking place on April 26th, 2010. I will be on stage with Raven Zachary[10] (creator of two Top 20 iPhone Apps, and Founder of iPhone DevCamps, and owner of the company that created the official Barack Obama iPhone app), Bill Dudney[11] (yes, the iPhone Development Guru and Best-Selling Author of iPhone SDK Development and Core Animation) Daniel Steinberg[12] (author of articles in O'Reilly's MacDevCenter and ADC, and also author of the Cocoa Programming[13] and Zero Configuration Networking[14] books) and Sumit Rai[15] (creator of the Deutsche Bank Application and Founder of Kulu Valley).

I look forward to meet you there in person! I will be blogging more about these events next year, as I am extremely honored about participating to both events, being 100% sure that they will be resounding successes.

Don't hesitate to stop me to say hi, or to e-mail to ask questions before the events. As usual, feel free to browse my previous presentations[16] on SlideShare.

Update, 2010-02-09: Unfortunately Bill Dudney will not be able to come, but Daniel Steinberg will replace him… with a surprise!

[8] http://www.trifork.com/
[9] http://oreilly.com/
[10] http://www.smallsociety.com/
[11] http://bill.dudney.net/roller/objc/
[12] http://www.oreillynet.com/pub/au/187
[13] http://oreilly.com/catalog/9781934356302/
[14] http://oreilly.com/catalog/9780596101008/
[15] http://www.kuluvalley.com/management-team
[16] http://www.slideshare.net/akosma/presentations

Reducing the Carbon Footprint

Adrian Kosmaczewski

2009-12-15

From the beginning, akosma software was created with the objective of providing its customers with high quality software in different platforms. However, we are aware of the complex social and environmental context we are living in, and the current and future challenges generated by our daily actions on the climate of the planet and the fate of our world.

In particular, the use of computers, smartphones, electronic devices and other gadgets clearly has a strong impact on the environment, which we should try to evaluate and reduce.

To address the issues of carbon emissions, akosma software makes a vow of respect of the environment, and takes the following decisions towards the reduction of its carbon emissions, which in turn have a strong human impact:

- We will promote and encourage the use of public transportation and/or bycicles over fossile fuel-powered private vehicles whenever possible.
- We actively reduce our energy consumption by turning off all lights and computers at night, instead of leaving them running in "sleep" mode. We also use energy-saving light bulbs and actively look for ways to reduce our energy consumption.
- We promote the use of telecommuting[1], using tools such as Google Docs[2], Skype[3] and instant messaging tools, which help people work together without having to be in the same room. While not always possible, our experience suggests that 60%

[1] http://en.wikipedia.org/wiki/Telecommuting
[2] http://docs.google.com/
[3] http://www.skype.com/

(some say even more[4]) of the normal work on our industry could be done from home. Which means less transportation and a more relaxed staff.
- We will always make sure that water taps in our offices don't leak and don't lose water. We try to reduce water losses and to enjoy responsibly every drop of it.
- We will avoid printing copies of electronic documents, unless explicitly required by third parties. We will use recycled paper as the base supply for our printing needs. The same goes by choosing not to buy paper copies of books or magazines, and instead to choose their electronic versions.
- We will subcontract out only to companies with a strong ecologic and ethic engagement[5], for example when dealing with hosted service companies[6] or any other supplier of services.
- We will properly dispose of, and recycle whenever possible, as much used material as possible, including, but not limited to, paper, aluminium, batteries, electronic devices, recyclable organic waste and other elements.

Through these simple steps, akosma software aims to reduce its environmental and social impact to the minimum, and we choose to disclose these steps publicly for raising awareness in other companies as well.

What are you doing to reduce your carbon footprint? Feel free to share your tips and tricks on the comments below.

[4] http://weblog.raganwald.com/2008/01/no-disrespect.html
[5] http://zerofee.org/
[6] http://www.site5.com/green-hosting/

Reflexions on the Software Business

Adrian Kosmaczewski

2009-12-15

There are basically two things you can do to earn a living when you write code:

- Consulting
- Products

When doing **consulting**, you write code, and somebody else owns it; you are blamed for its bugs, rarely praised for its benefits, and usually you only sell one copy of your work. When working on **products**, you write code, and you actually own it; you can brag about it on your blog without pissing anyone, and if you are lucky you sell as many copies of it as you want, all for basically the same production cost.

Now, here's an insider tip: if your objective is living a nightmare, tearing yourself apart and swear never touching a keyboard again, choose option 1. If your objective is enjoying a healthy life, making money and living long and prosper, choose option 2.

This fact is explained by economists as a "diseconomy of scale"[1]: this means that fixed costs are very low relative to variable costs, which means that the cost of creating a new copy of your finished product is virtually zero. You only have to invest in the building, not on the replication. Actually this is not 100% true, because you should spend on marketing anyway, and you might as well add new features on the way, but the truth is that well-run software companies make more money than drug dealers, and guess what: software is an activity usually considered legal.

However, there is a tacit consensus in Switzerland, apparently, by which there can't be successful companies doing software in this side of the world. And most companies choose option 1 above. Which has interesting side effects.

Consulting

Consulting, just like the airline industry, succeeds in one particular point: it pisses off everyone involved in it. Let's be frank; clients are seldom happy of the end result, while consultants have to deal with

[1] http://www.softwaremetrics.com/se.htm

horrible working environments (read: open spaces). The only ones actually enjoying this market are the (usually non-technical) owners of consulting companies, who take pride in the fact of selling an "expert" to a company for around CHF 1000 per day (much more in the case of SAP), while they pay less than CHF 300 to the same consultant. The remaining 700 go to "operational costs", of course, including the bonuses paid to managers of these companies on the backs of the workers.

Welcome to "Capitalism 101". You have to afford that new Porsche somehow.

Not only are consultants screwed from day one, with the typical speech of "we are a human company, people is our first priority", they also get fired first whenever the market shrinks. They have to beg for training and to be sent to conferences, while their managers go to corporate retreats in Davos or Zermatt. Heck, sometimes consultants even have to ask for a proper computer to do their jobs, or are cynically asked to use their own personal equipment.

Oh, and consultants have to fill timesheets, and get punished if they don't do it. Timesheets are worth an article of their own, in the sense that they are only used as command-and-control tools, and not, as one would think, as the basis for future estimations of upcoming projects. Timesheets are just black holes of information, where you might as well log 8 hours in the "whatever" category and nobody would really care. And estimations are usually done by your non-technical boss, anyway, so screw those historical data.

(Sometimes consultants not only have to fill their employer's timesheet, but also the customer's. I remember that at one time I had to fill 3 different timesheets. I could easily spend 2 hours a week making sure everything was right and coherent. And no, there wasn't any "timesheet filling" entry in the timesheet software. And even worse, timesheet software - web based or not - usually sucks big time.)

OK, I'm probably being unfair here. There are a couple of benefits to being a consultant. I suppose. I hope. But this is not my point.

Products

As shown, the consulting landscape does not look very promising; on one side, many companies try to eat a small consulting market using the same shitty practices. On the other side, thankfully, there are companies[2] who have understood that you can earn a very decent living by creating a nice product and selling licenses (or subscriptions) of it.

As previously, there are interesting side effects to choosing this strategy:

[2] /blog/roundup-of-swiss-companies-writing-mac-apps/

- Creating products has the ultimate goal of generating a steady income stream. This frees up energy and resources in your team to build new products, which generate more revenue, which you can spend creating new products… and so on and so forth. You get the idea.
- Having to maintain a few products means that you can afford knowing its quirks by heart; you don't have to context switch from project to project like most consultants do, and you can continuously fix bugs and add new features to it. You feel like the product is your child, and you help it grow and become stronger, more resilient, more powerful. Which helps you sell more copies, etc, etc (see the previous point).
- Google's much touted "20% project time" becomes, in the case of owning your own products, a "100% project time". You enter a state of continuous creation. You don't have to explain your choices to a non-technical (read: incompetent) boss: you respond, at most, to what your market demands (read: your customers).
- You can create a product suite; the synergy created from one product to the other might suffice to drive sales up of both products all by itself.
- You can have a direct contact with your clients, answering their requests and problems, instead of relying on a (usually non-technical) man-in-the-middle strategy.
- You acknowledge the fact that 8 hours of coding work is an illusion. I know no developer capable of sitting for 8 hours in front of a computer and writing coherent code, which is what most consultants are expected to get for CHF 1000 per day (the customer doesn't usually know that a consultant only gets 30% of that sum). A maximum of 5 or 6 hours of pure concentration is already a big win, and the rest should be spent doing paperwork, playing Wii Sports or doing the groceries. Freeing your mind helps you have more ideas, which in turn become products that generate new revenue streams. When you are in consulting mode, you cannot have this liberty. Actually you have no liberty at all.
- You can have a real quality strategy. I know no consulting firm which really pays attention to quality (even if most fill their mouths with the Q word). Refactoring, unit testing, user testing, writing requirements and specs are just nonexistent tasks in most consulting companies. When you are creating products, you can take time to do them with the depth that you want; and actually, you do it, and you enjoy it.

Again, I'm really being unfair here. I am concentrating maybe too much into this "circle of virtue" called "product -> revenue -> freedom -> product -> rinse and repeat". Things are never that easy; when you create a product, you have to choose a platform, find a market for it, invest in the creation part, advertise it, maintain it, support your customers, update your website, burn the CD-ROMs, write

in your blog, test your product in the next version of the operating system (or browser), fix that weird Unicode bug, set up the eShop for selling your product, troubleshoot PayPal issues, add entries to the FAQ, participate in trade shows, send demos to magazines, fix the damn coffee machine, and many, many other things.

However hard it might seem, the underlying truths are fundamental: when you own the product, your commitment to quality and your enthusiasm will be unparalleled. And your code will be better just because of that.

Conclusion

I would say that consulting is a viable option to start up, as a short term strategy. There's a lot of demand for custom software out there, and using your brain to generate cash that way can be used as a quick entry point to bootstrap your own company.

However, in the medium and long term, the only viable strategy for sustained growth in the software industry is the creation and sale of software products. This is the only way to create true value in your own company, helping you create a healthy environment for your staff, fostering creativity, engaging customers with a real experience, and creating a win-win situation for you and your customers.

Of course, creating and managing a product requires skills and objectives which are not the same as your usual consulting project; this is the reason why most consulting companies fail when trying to jump on a product mindset. This will be the subject of a future article.

10 Things Every iPhone App Designer Should Know

Adrian Kosmaczewski

2009-12-21

Design is a fundamental part of iPhone app development. It is, without any doubt, the difference between a crappy and a great application. It can be the discriminating factor of life and death on the App Store, and the competition is brilliant and strong. The best teams, like Tapbots[1], Sophiestication[2] or Jilion[3] have understood that design is part of the process: not a nice to have, and certainly not an afterthought.

[1] http://tapbots.com/

[2] http://www.sophiestication.com/

[3] http://jilion.com/

[4]

However, this does not mean that designers coming from a web or print background are ready to tackle design projects for the iPhone ipso facto. I've seen too many horrors so far[5], and that is why I am making public this list of tips and tricks that I keep repeating to all the designers I work with.

1. **Read the iPhone design guidelines.** Thoroughly. Twice or

[4] http://www.flickr.com/photos/thetechbuzz/3709438002/
[5] /blog/dirty-little-secret/

even better, three times until you know them by heart. Depending on the budget and the timeline, it might be impossible to create custom components or to achieve all your fancy animations and effects; in that case, you should stick to the UIKit widgets catalog for your designs, including all of their limitations. Once you've read the guidelines, buy these books and read some more:

- Programming the iPhone User Experience[6], by Tody Boudreaux;
- Best iPhone Apps[7], by Clark Josh;
- iPhone User Interface Design Projects[8] by David Barnard, Joachim Bondo and others.

2. **The iPhone is more like a motion picture than a static image.** Even the most basic applications feature animation, beautiful scrolling and swiping and pinching. Thus, that gorgeous Photoshop file is not enough; you will have to explain to me your idea, frame by frame, like if you were writing a **screenplay**; every animation, every step, every transition must be as documented as possible for the developers to be able to bring your design to life.
3. **Fingers are bigger than mouse pointers.** The minimum size for an area that should be tapped is 44 pixels wide. Smaller buttons or icons are simply unusable. And since we're talking about sizes, the status bar is 20 pixels high, toolbars and navigation bars are 44 pixels high.
4. **The iPhone has a limited set of fonts.** Not all the fonts you have on Photoshop are available on the iPhone, and at least so far, there is not an easy way to integrate external fonts in an iPhone application. Grab a copy of RooiFonts[9] and learn which fonts are available on the iPhone. And avoid Marker Felt altogether[10]. Please.
5. **The four golden screen sizes:**

   - Portrait without status bar: 320 x 480
   - Portrait with status bar: 320 x 460
   - Landscape without status bar: 480 x 320
   - Landscape with status bar: 480 x 300

6. **Think PNG.** Whenever you have to provide graphics for the

[6] http://www.amazon.com/Programming-Iphone-User-Experience-Boudreaux/dp/0596155468%3FSubscriptionId%3D0F0YTN83N46JSX6KDT02%26tag%3Dakosmasoftwar-20%26linkCode%3Dxm2%26camp%3D2025%26creative%3D165953%26creativeASIN%3D0596155468

[7] http://www.amazon.com/Best-Iphone-Apps-Discriminating-Downloaders/dp/059680427X%3FSubscriptionId%3D0F0YTN83N46JSX6KDT02%26tag%3Dakosmasoftwar-20%26linkCode%3Dxm2%26camp%3D2025%26creative%3D165953%26creativeASIN%3D059680427X

[8] http://www.amazon.com/iPhone-User-Interface-Design-Projects/dp/1430223596%3FSubscriptionId%3D0F0YTN83N46JSX6KDT02%26tag%3Dakosmasoftwar-20%26linkCode%3Dxm2%26camp%3D2025%26creative%3D165953%26creativeASIN%3D1430223596

[9] http://rooifonts.com/

[10] http://daringfireball.net/2008/11/iphone_likeness

iPhone, remember that PNG is the best solution. It supports transparency, lossless compression, lots of colours, and the iPhone can optimize them even further. By the way, every app needs a “Default.png” file (shown during startup) of one of the four golden screen sizes enumerated above. And an “Icon.png” file, 57 x 57 pixels wide.

7. **Create your icons using the biggest possible resolution you can.** The App Store requires you to provide a 512 x 512 TIFF file, at 72 DPI, but if your application is successful, they might ask for a higher resolution file for a commercial. Your best idea then is to design your icon as if it were to be printed in a huge banner[11].
8. **The status bar on top of the application can be removed altogether**. Unfortunately it can’t be customized beyond a small set of options, and basically you can only have it gray, black or translucent. You can apply a bit of color to the status bar by having a coloured view underneath, but it might not look the way you want.
9. **The iPhone OS provides some functionality for free**, including all the widgets and some more obscure behaviour, for example, tapping the status bar makes lists scroll to the top, and tapping a tab bar item makes the navigation return to the root controller. Some other things, like Convertbot-like “scrollwheel menus” are not part of the toolkit and might take a bit longer to create.
10. **Photoshop is not enough.** Use a paper-based design sketchbook[12], preferably one with a stencil, and draw your application manually. This will help you figure out features and interaction without committing to a visual design. Then ask your nearest developer for the iPhone SDK, install it and play with Interface Builder. I said play: don’t try to understand everything you see, because it’s a rather complex tool. But you can drag and drop components, and the inspector allows you to change styles and move things around. You will learn a lot, and most importantly, you will see things through the eyes of a developer.

I’d like to hear more tips and tricks. Feel free to leave them in the comments below!

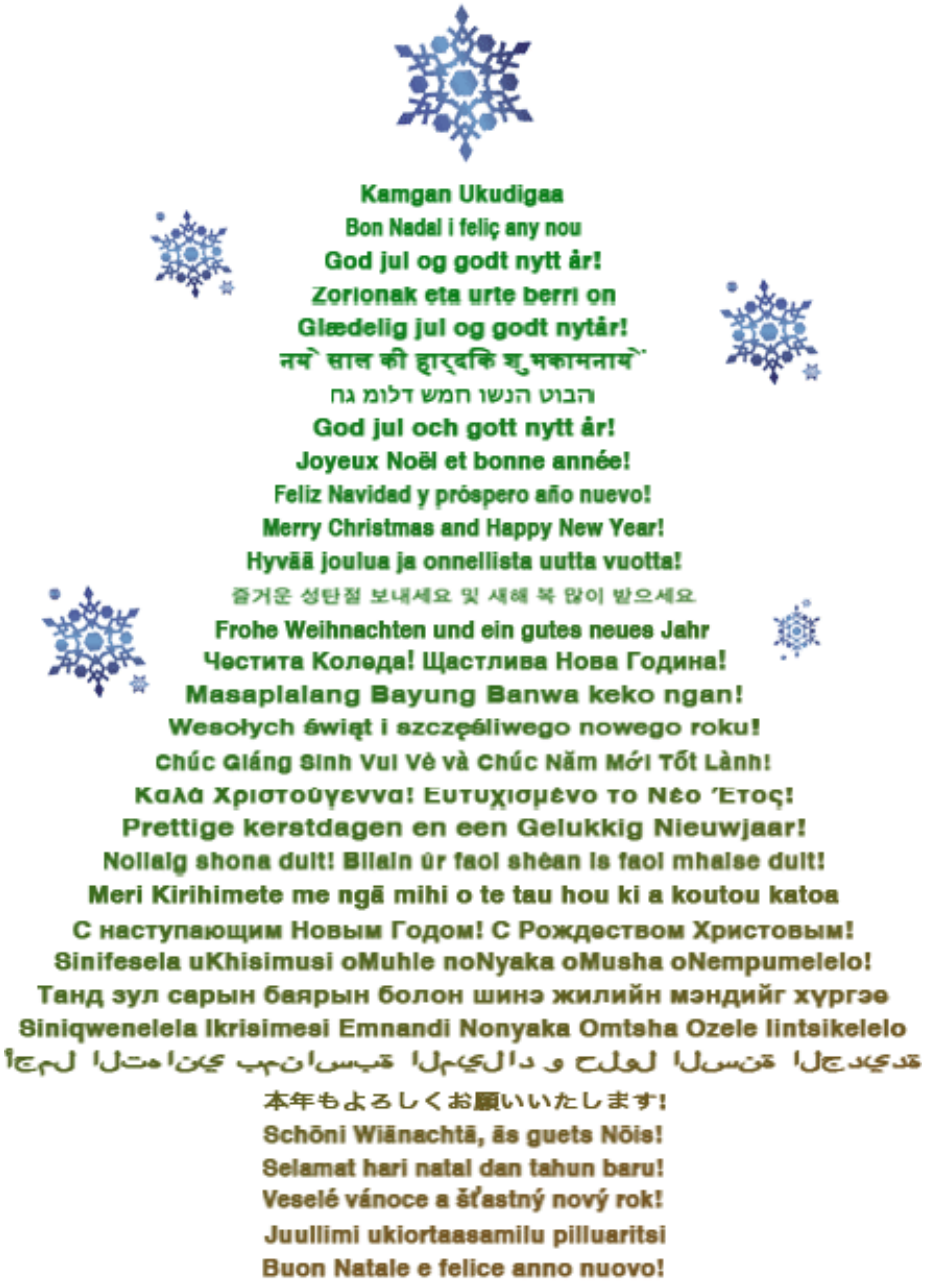
[11] http://www.flickr.com/photos/39995160@N03/3683399203/
[12] /blog/roundup-of-iphone-app-sketchbooks/

Season’s Greetings

Adrian Kosmaczewski

2009-12-23

Whenever you are, whichever language you speak akosma software wishes you happiness and health for 2010!



Thanks to Claudia[1] for the artwork and to Omniglot.com[2] for the phrases

[1] http://twitter.com/claukosma
[2] http://www.omniglot.com/language/phrases/christmas.htm

Mention on Patrice Neff's blog

Adrian Kosmaczewski

2009-12-23

"Yahoo! Blueprint for mobile sites"[1] in Patrice Neff's blog.

> The past few days I created a mobile site[2] for Memonic[3]. During this I made use of some of the newly found knowledge from the past Webtuesday[4] which Adrian Kosmaczewski[5] was kind enough to share. See also my notes about iPhone web development[6] from that event.

[1] http://weblog.patrice.ch/2009/12/21/yahoo-blueprint.html
[2] http://m.memonic.com/
[3] http://www.memonic.com/
[4] http://www.webtuesday.ch/meetings/20091208
[5] http://kosmaczewski.net/
[6] http://www.memonic.com/user/pneff/set/all/id/1aeR

Kevin Smith iPhone Application

Adrian Kosmaczewski

2009-12-23

I've just been notified by the nice guys from DenVog[1] that the Kevin Smith iPhone application[2] includes code from my Asynchronous UITableView sample[3], posted earlier this year. The app also features the MGTwitterEngine from Matt Gemmell[4], too.

[1] http://www.denvog.com/

[2] http://www.denvog.com/iphone/KevinSmith/index.html

[3] /blog/asynchronous-loading-of-images-in-a-uitableview/

[4] http://twitter.com/mattgemmell/status/6896941486

Thanks for the credits guys! I’m glad my code has been helpful to you.

Del.icio.us to Wordpress

Adrian Kosmaczewski

2009-12-28

I've just uploaded a new project on Github called delicious_wp[1]: it's a small Ruby script that simply fetches the items stored in del.icio.us the previous week and creates a blog post with them. You can set up a small cron job to execute this script every week, which is what I've done for this blog :) I know del.icio.us has a similar feature integrated, but it executes daily, instead of weekly, which is what I wanted.

To use it, just clone the repository, copy the config.yaml.sample file as config.yaml and edit its values inside. Run the script and voilà! A new blog post entry with your del.icio.us bookmarks.

The script can also be helpful to those wondering how to use the XML-RPC interface of WordPress from a Ruby script, or how to use the Net::HTTP library to consume a REST API.

```
def get_delicious_bookmarks
  # Connect to delicious and get updates
  http = Net::HTTP.new(DELICIOUS_SERVER, DELICIOUS_PORT)
  http.use_ssl = true
  req = Net::HTTP::Get.new(DELICIOUS_DATES_PATH)
  req.add_field("User-Agent", DELICIOUS_USER_AGENT)
  req.basic_auth username, password
  response = http.request(req)
  results = response.body
```

```
def post_to_wordpress(title, text)
  entry = {
    :title => title,
    :description => text
  }
  # Connect to Wordpress using the XML-RPC interface
  blog = XMLRPC::Client.new(server, path, port)
  blog.call("metaWeblog.newPost", blogid, username,
                              password, entry, true)
```

Enjoy! As usual, the code is released with a BSD license.

[1] http://github.com/akosma/delicious_wp

EasyTableView for iPhone Prototyping

Adrian Kosmaczewski

2010-01-17

Our first blog post of 2010 presents a tool that we've been using internally and that might be useful for other people.

As you might have experienced, prototyping applications in Interface Builder is fine as long as you don't deal that much with UITableViewController instances. Prototyping user interfaces with tables and navigation is not as easy as drag-and-dropping some components, and you usually have to implement a small UITableViewController subclass from scratch yourself, including some kind of navigation logic.

Enter EasyTableView[1], a set of classes inheriting from UINavigationController and UITableViewController, ready to use in your own projects, to quickly simulate interfaces using navigation and tables. I've been using this code in several prototypes, and it allows me to quickly craft navigation-based applications loading several different tables. The AKOEasyTableViewControllerDelegate protocol allows other classes to be notified of taps on cells or on accessory buttons.

The code is available, as usual, in Github[2] with a liberal BSD license.

The code snippets below show how to use the classes:

```
NSArray *data = [NSArray arrayWithObjects:@"first", @"second", @"third", nil];

AKOEasyTableViewController *controller = nil;
controller = [[AKOEasyTableViewController alloc] initWithStyle:UITableViewStyle
controller.title = @"Easy Table View";
controller.delegate = self; // AKOEasyTableViewControllerDelegate...
controller.dataSource = data;
controller.autoDeselect = YES;
controller.accessoryType = UITableViewCellAccessoryDisclosureIndicator;
_navigationController = [[AKOEasyNavigationController alloc] initWithRootViewCo
_navigationController.dataSequenceFileName = @"Sequence";
```

With the code above, the freshly created AKOEasyNavigationController will load the file "Sequence.plist", which itself references other plist files in your project.

Data can be loaded in code (as NSArray or NSDictionary instances) or using external plist files, through the dataSourceFileName property of the AKOEasyTableViewController class. The formats of the data in the plist files is the simplest you could imagine:

```
<?xml version="1.0" encoding="UTF-8"??>

<plist version="1.0">
<array>
    <string>Roger Federer</string>
    <string>Rafael Nadal</string>
    <string>Novak Djokovic</string>
    <string>Andy Murray</string>
    <string>Juan Martin Del Potro</string>
    <string>Andy Roddick</string>
    <string>Nicolay Davydenko</string>
    <string>Fernando Verdasco</string>
    <string>Robin Soderling</string>
    <string>Jo-Wilfred Tsonga</string>
    <string>Fernando González</string>
    <string>Radek Stepanek</string>
    <string>Gael Monfils</string>
```

[1] http://github.com/akosma/EasyTableView
[2] http://github.com/akosma/EasyTableView

```
    <string>Marin Cilic</string>
    <string>Gilles Simon</string>
</array>
</plist>
```

The plist files can also be NSDictionary instances, with a “title” NSString key, and a “values” key of type NSArray; in this case, the controller is shown with several different groups. AKOEasyNavigationController requires the AKOEasyTableViewController class, but you can use this last one independently as well.

Feel free to clone the project[3], use it in your own projects and send me any feedback you might have. Enjoy!

[3] http://github.com/akosma/EasyTableView

The Very Quick and Sloppy Guide to Argentine Rock

Adrian Kosmaczewski

2010-01-18

From Wikipedia[1]:

> The Argentine rock movement was truly one of the first non-English forms of rock to be commercially successful outside its own nation.

Argentine rock, which was the first kind of rock in Spanish ever to emerge in either Spain or Latin America, has a "founding trilogy" in 1967 with three mythical bands:

- Los Gatos[2] ("The Cats"), similar to the Beatles, they disbanded in the 60's;
- Manal[3], more of a blues / hard rock thing, first aligned with the Stones and later aligned with the heavy metal movement until the 80's;
- Almendra[4] ("Almond") which was the real start of argentine progressive rock, also disappeared in the 60's.

The founder of Almendra is the ENORMOUS, GIANT, KING OF PROGRESSIVE, Luis Alberto Spinetta[5], one of the best musicians of the 20th century, without any doubt, with over 45 years of a non-stop career. He's still playing occassionally, and in 1973 he recorded with a later band ("Pescado Rabioso", "Rabid Fish" in Spanish) the album "Artaud[6]" which is regarded as one of the highest moments of argentine progressive rock, and one of the best albums of its time.

[1] http://en.wikipedia.org/wiki/Argentine_rock
[2] http://en.wikipedia.org/wiki/Los_Gatos_(band)
[3] http://en.wikipedia.org/wiki/Manal
[4] http://en.wikipedia.org/wiki/Almendra_(band)
[5] http://en.wikipedia.org/wiki/Luis_Alberto_Spinetta
[6] http://en.wikipedia.org/wiki/Artaud_(album)

Above: Luis Alberto Spinetta (left) and Charly García in the ’80s

Listen to track #7 “Bajan” (“They are going down”) and #9 “Las Habladurias del Mundo” (“The Gossip of the World”). Those two tracks are the best IMHO, but in the good progressive fashion, the album is a highly conceptual one with a central story centered around Antonin Artaud[7]’s poems, that can be listened from start to end like a an opera.

Spinetta played not long ago in the Colón Theatre[8], one of the most important and biggest opera houses of the world, which is said to have “one of the five best acoustics in the world”...! I’ve been there twice. It’s unbelievably beautiful.

Sumo[9]: this band was formed in 1981 by Luca Prodan[10], an Italian/English guy who went to Argentina to escape his heroin addiction, but died of cyrrhosis and cocaine in 1988, after having changed the argie music forever. He was the only one who dared sing in English in Argentine radios during the Falkland’s war.

His death was so terrible to argentines that graffittis saying “Luca Not Dead” still appear (written in English) in Buenos Aires. His influence is absolute. All the bands now say they were influenced by Sumo. I still remember the newspapers announcing his death. I was in secondary school at the time. All the kids were sad, even crying.

[7] http://en.wikipedia.org/wiki/Antonin_Artaud
[8] http://en.wikipedia.org/wiki/Teatro_Colón
[9] http://en.wikipedia.org/wiki/Sumo_(band)
[10] http://en.wikipedia.org/wiki/Luca_Prodan

[11]

I recommend their album Divididos por la Felicidad[12] (“Divided by Happiness”). Start with the all-time-classic #1, “La Rubia Tarada” (“The Dumb Blonde”), a rant against the trendy first class of Buenos Aires in the 80’s, and then move to #7, “Mejor no Hablar de Ciertas Cosas” (“Better not to talk about some things”). These two songs are simply, classics.

After Prodan died, Sumo split into two bands: Divididos (“Divided”) and Las Pelotas (“The Balls”). The first is more commercial and high profile, the second more progressive and underground. Divididos is called the “La Aplanadora del Rock” (“The Leveller of Rock”) given that they offer mega-concerts (at 100’000 people piece) lasting over 4 hours of pure rock.

Divididos’ album Narigón del Siglo[13] (“Big-nosed-guy of the Century”) was recorded at Abbey Road studios, and is particularly representative of Divididos. This is really, really a rock band, in the purest sense of the term. My preferred tracks are #6 “Elefantes en Europa” (“Elephants in Europe”), which will explain you the “The Leveller of Rock” nickname…! And #11 “Pasiones Zurdas Derechas” (“Right-wing Lefty Passions”). Amazing rock. Really puts you in good mood.

From Las Pelotas’, I recommend Mascaras de Sal[14] (“Masks of Salt”). Much more studied, thoughtful, very interesting and progressive. I love it. I recommend “Senderos” (“Paths”), track #1. But the whole album is simple IN-CRE-DI-BLE.

Soda Stereo[15]: (“Stereo Sparkling Water”) is one the best selling argentine acts of the last 30 years. They are the incarnation of argentine pop, simple, and pure. They started in 1981 and finished in 1997 with a mega concert of 3 days in the largest football stadium in Buenos Aires[16]. Listen to their 20 Greatest Hits[17] album, for your listening pleasure: they are REALLY good, timeless kind of good. Start with

[11] https://www.flickr.com/photos/70773894@N00/328122048/
[12] http://itunes.apple.com/ch/album/divididos-por-la-felicidad/id321550458
[13] http://itunes.apple.com/ch/album/id321864763
[14] http://www.rock.com.ar/discos/1/1068.shtml
[15] http://en.wikipedia.org/wiki/Soda_Stereo
[16] http://en.wikipedia.org/wiki/Estadio_Antonio_Vespucio_Liberti
[17] http://itunes.apple.com/ch/album/originales-20-exitos/id187445222

tracks #3 “Nada Personal” (“Nothing Personal”) which is really representative of their 80’s, and then go to tracks #15 “Lo Que Sangra - La Cúpula” (“What Bleeds - The Cupola”) and #14 “Un Millón de Años Luz” (“One Million Light-Years”), the latest a much more mature and elaborated song, considered the most beautiful track of the band. Really my all time favorites.

Finally, Bersuit Vergarabat[18] (or simply Bersuit, a name that doesn’t mean anything, really :) simply put, the most important band of the past 20 years. And still alive and kicking. They are very good. Capables of everything and anything: ska, rock, tango, pop, samba, murga[19], etc. Bersuit is one of those bands that 40 years from now will have streets named after them. They are really the biggest milestone in Argentine music since Sumo.

Their most interesting studio album is Hijos del Culo[20] (“Sons of the A**” :) Listen to #12 “Negra Murguera” (“Black Woman from the Murga”) and #2 “La del Toro” (“The Song of the Bull”). The whole album goes from one style to the other, their lyrics are hilarious and at the same time terrible, they really sing the Argentine of today.

To close the loop, here’s the main singer of Soda Stereo (Gustavo Cerati, on the left) with Spinetta himself (at the right) singing “Bajan” from Pescado Rabioso (which I mentioned above) live:

And the argentine rock fan watches in awe, reverence, wonderment, admiration, as two epochs meet, greet and sing.

This video is a cornerstone moment in the history of argentine rock. I’m almost in tears.

[18] http://www.bersuit.com/
[19] http://en.wikipedia.org/wiki/Murga
[20] http://www.rock.com.ar/discos/0/165.shtml

Talking at the Skills Matter Meeting in London Next Week

Adrian Kosmaczewski

2010-01-27

Preparing the path for the amazing iPhone Dev Days 2010[1] in April, next week I'll be giving some tips and tricks about iPhone development[2], in London!

This will happen in the Skills Matter[3]'s meeting, at the The Skills Matter eXchange[4], 116-120 Goswell Road, London EC1V 7DP, on Tuesday, February 2nd, at 18:30.

I hope to see you there! The event is free so don't hesitate to come by.

[1] http://iphonedevday.com/iphone-london-2010/

[2] http://skillsmatter.com/event/os-mobile-server/tips-and-tricks-on-building-successful-iphone-apps

[3] http://skillsmatter.com/

[4] http://skillsmatter.com/location-details/os-mobile-server/602/96

nib2objc featured in Softpedia

Adrian Kosmaczewski

2010-01-29

As the title suggests: check out the official nib2objc page in Softpedia[1]!

[1] http://mac.softpedia.com/get/Developer-Tools/nib2objc.shtml

Skills Matter Presentation

Adrian Kosmaczewski

2010-02-03

Yesterday's presentation[1] in the Skills Matter[2] meeting is available in Slideshare[3]!

Update, 2010-02-05: The kind guys at Skills Matter have also published the video of the whole conference; check it out![4]

[1] http://skillsmatter.com/event/os-mobile-server/tips-and-tricks-on-building-successful-iphone-apps

[2] http://skillsmatter.com/

[3] http://www.slideshare.net/akosma/tips-tricks-for-iphone-application-development

[4] http://skillsmatter.com/podcast/os-mobile-server/tips-and-tricks-on-building-successful-iphone-apps

iPhone Dev Day in Geneva on April 28th

Adrian Kosmaczewski

2010-02-10

Those of you who have been following me for long might remember an article I wrote in my personal blog[1] two years ago (in French, translated here):

> Why can't we have conferences like this one[2], with mini events like this one in parallel[3] in the French-speaking part of Switzerland? Or even like this one[4]? Or like this other one[5]! Or that one[6]!

For all of you non-French speakers, this blog post is about a dream, about bringing to Switzerland quality speakers, for a world-class conference about technology. Something that this part of the country, at least, was lacking.

Well, my dream has come true: following the success of the 2009 Zürich event[7], and 2 days after the 2010 London edition[8], I hereby proudly announce the Dev Days for iPhone 2010 in Geneva[9], on Wednesday, April 28th, to take place in the Geneva Business Center[10] in Petit-Lancy. The conference is co-organized by Trifork[11], Hortis[12] and akosma software, and sponsored by none other than

[1] /blog/pourquoi-pas/
[2] http://www.meshconference.com/
[3] http://www.meshconference.com/meshu/
[4] http://futureofwebapps.com/
[5] http://www.web2con.com/
[6] http://www.futureofwebdesign.com/
[7] http://iphonedevday.com/suisse-2009/
[8] http://iphonedevday.com/iphone-london-2010/
[9] http://iphonedevday.com/iphone-geneva-2010/
[10] http://www.geneva-business-center.com/
[11] http://trifork.ch/
[12] http://www.hortis.ch/

O'Reilly[13].

The conference is built around a double track, one for business people, the other for developers. As for speakers, we are immensely proud and happy to have on stage Raven Zachary[14] (creator of the Obama iPhone app back in 2008, owner of Small Society[15] and one of the most important figures in the iPhone business world in the USA), Daniel Steinberg[16] (author of Cocoa Programming[17] and Zero Configuration Networking[18], and frequent writer in the ADC[19] site or O'Reilly's MavDevCenter[20]), and Neelan Choksi[21] (founder of Lexcycle[22], the company behind the Stanza iPhone application).

Faithful to the "think global, act local" meme, the event will also feature business presentations of local interest, highlighting the thriving ecosystem built around the iPhone in Switzerland, including presentations Frédéric Layani[23] and Nicolas Marion[24] from Tag Heuer[25], and Jérôme Layat[26] from the Hortis team[27], who have recently collaborated in publishing an astonishing iPhone application[28].

In any case, a world-class, unique event, without any doubt the most important iPhone-related event to happen in Switzerland in 2010. We hope to meet you there and share some interesting insight about the iPhone business!

For any inquiries about the event, don't hesitate to contact Geeta Schmidt (+ 45 87 32 87 87, gsm@trifork.com[29]) or myself[30].

[13] http://www.oreillygmt.co.uk/
[14] http://www.rinzai.com/ravenzachary.html
[15] http://www.smallsociety.com/
[16] http://www.oreillynet.com/pub/au/187
[17] http://oreilly.com/catalog/9781934356302/
[18] http://oreilly.com/catalog/9780596101008/
[19] http://developer.apple.com/
[20] http://macdevcenter.com/
[21] http://www.linkedin.com/in/neelan
[22] http://www.lexcycle.com/
[23] http://iphonedevday.com/iphone-geneva-2010/speaker/Frederic+Layani
[24] http://iphonedevday.com/iphone-geneva-2010/speaker/Nicolas+Marion
[25] http://www.tagheuer.com/
[26] http://iphonedevday.com/iphone-geneva-2010/speaker/Jérôme+Layat
[27] http://www.hortis.ch/
[28] http://itunes.apple.com/en/app/monaco-v4/id341640335?mt=8
[29] mailto:gsm@trifork.com
[30] /about/

The Dawn of Couch Computing

Adrian Kosmaczewski

2010-02-17

I haven't blogged about the iPad until now on purpose; I wanted to let the waters calm down, as January's announcement was certainly one that left nobody indifferent.

The iPad is a disruptive device. Not in the sense of groundbreaking, because there is nothing on the iPad that you couldn't find in other devices. I mean disruptive in the sense of creating a new blue ocean[1], a new category of devices where the interaction with the machine is done through a completely different kind of approach. Apple is, once again, redefining the word "personal computer", setting itself apart from the competition, and making us drool all over again.

The iPad is the dawn of couch computing[2], with all what this phrase means.

The critics to the iPad are doing the same level of noise you can hear whenever Apple releases a new product. Remember: it all started

[1] http://www.blueoceanstrategy.com/

[2] http://twitter.com/0xced/status/8915818053

with Dvorak critisizing the first Macintosh because of the mouse[3]. Then it continued with the iMac not having a floppy disk[4]. Then it was the iPod and its hard drive or its lack of a color screen. Finally it was the iPhone, which has generated as many critics as sales. All what is written around the iPad follows the same pattern.[5]

To help to filter the noise produced by all the opinions, keep in mind the following elements:

- We geeks are a very small portion of the human population. A noisy one, but very small anyway. Our opinions are biased and most of the time are not the same as the average user. The iPad is not a device for geeks. This is a device for common people, who want a lightweight computer that starts fast, can be used on a couch, looks beautiful and can even be used to create small documents.
- The average user wants to hold in their hands whatever new gadget seen in the movies. One of the best reviews of the iPad I've read so far[6] stated that the device is a step in a trend to bring gadgets found in "Star Trek", "Star Wars" or "2001" to the market.
- The iPad is the ultimate simplification of the user experience. This device completely isolates the user from the underlying hardware. The user interface is the computer. No file system[7], no files, no folders, just apps, documents within those apps, and as much multimedia as possible: videos, images, sound. The same hardware is used for both primary input and primary output.

Is there a use case for the iPad? You are pretty damn sure there is: right now I'm typing this blog post on MarsEdit on a train. I use my MacBook Pro for that. But I know I would use an iPad instead if I had it. I know designers planning to buy some iPads to showcase their portfolio in them. Not to mention on-demand business opportunities, gaming, social networking, and much more.

Couch computing means more than dumb consumption of media: it is also a whole new relationship with the way humans create and share information. The iPad is the first real electronic canvas, and its über-simplification of the user interface, together with an übersimplification of the distribution of software, will create a whole new market. A whole, new, different paradigm. The iPad is not a computer, it is not a netbook, it is not a laptop. It's an iPad. The user experience patterns will be different, the use cases are yet unforeseen.

[3] http://bethesignal.org/blog/2009/01/13/qotd-john-c-dvorak/

[4] http://www.everymac.com/systems/apple/imac/faq/imac-g3-pci-based-chrp-rom-why-no-floppy.html

[5] http://www.fastcompany.com/article/quiz-are-these-comments-about-2001-ipod-or-2010-ipad

[6] http://www.boingboing.net/2010/02/02/arthur-c-clarks-2001.html

[7] http://twitter.com/gil_les/status/9193995097

I have been contacted by many clients already to take their iPhone apps to the next level, and some ideas are just incredible. In any case, akosma software is your partner of choice for your next iPad application! Contact us, we'd love to hear about your next idea.

PS: by the way, did you know that in the next Dev Days for iPhone in London[8] and Geneva[9] (if the iPad NDA drops) Daniel Steinberg[10] will talk about the iPad instead of Core Data?

[8] http://iphonedevday.com/iphone-london-2010/

[9] http://iphonedevday.com/iphone-geneva-2010/

[10] http://iphonedevday.com/iphone-geneva-2010/speaker/Daniel+Steinberg

Mobile Marketing: Branding Seminar in Zürich – Save 15%!

Adrian Kosmaczewski

2010-02-18

Aberla[1] is organizing a Mobile Marketing & Branding Seminar[2] in Zürich, on Tuesday, April 27th, 2010, from 1300 to 1800, at the Renaissance Hotel, Thurgauerstrasse 101, Zurich[3].

This event will feature presentations from world-class thought leaders, about the strategies required to succeed in the world of mobile applications – not only for the iPhone but also for Android, BlackBerry, and other platforms!

[1] http://www.aberla.com/

[2] http://www.aberla.com/events/tabid/59/vw/3/itemid/6/d/20100427/language/de-de/Mobile-Marketing--Branding-Seminar.aspx

[3] http://maps.google.com/maps?f=q&source=s_q&hl=en&geocode=&q=Thurgauerstrasse+101,+Zurich,+8152+Switzerland&sll=50.377975,2.981902&sspn=12.061757,20.126953&ie=UTF8&hq=&hnear=Thurgauerstrasse+101,+Glattbrugg+8152,+Z%C3%BCrich,+Switzerland&t=h&z=16

- Sumit Rai, Founder of Kulu Valley[4];
- Harald Horber, Plattform Manager Mobile in the CFF / SBB[5];
- Raven Zachary, founder of Small Society[6] and creator of the iPhoneDevCamps[7];
- KC MacLaren, Manager of Retail Consumer Technology for Starbucks[8].

The inscription is open![9] As a special offer, readers of this blog can get 15% off the ticket price using the code **akosma**. Register now![10]

[4] http://www.kuluvalley.com/
[5] http://sbb.ch/en/
[6] http://smallsociety.com/
[7] http://www.iphonedevcamp.org/
[8] http://www.starbucks.com/
[9] http://www.amiando.com/aberla_2.html
[10] http://www.amiando.com/aberla_2.html

Senbei, a Fat Free CRM iPhone Client

Adrian Kosmaczewski

2010-02-24

I'm a happy user of Fat Free CRM[1] by Michael Dvorkin[2]. I've been using it since the start of akosma software to keep track of my projects, clients, prospects and even as a simple to-do list. It helps me as a "corporate memory", tracking every interaction I have with my clients and partners (phone calls, meetings, new projects, quotations, etc).

As I said[3] many times before[4], if you are not using Fat Free CRM, you should. A CRM, I mean, not a CMS :)

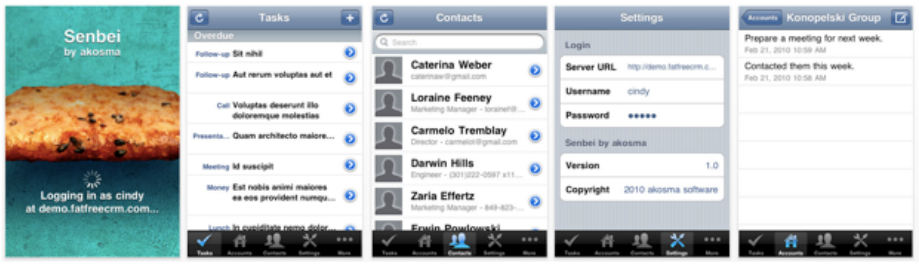
[5]

But, as an avid iPhone user, I was lacking the ability to use all of this information while I was in the road. I wanted to be able to access my "company wide" address book, check my tasks for the day, or add comments about the latest meetings right from my iPhone. So that is why, in cooperation with Michael and the marketing geniuses of Zerofee[6] (who suggested the name and contributed the icon), I've written Senbei[7], a small, lightweight iPhone client for Fat Free CRM.

[1] http://www.fatfreecrm.com/
[2] http://twitter.com/mid
[3] http://twitter.com/akosma/status/5682081696
[4] http://twitter.com/akosma/status/4678792954
[5] http://itunes.com/apps/senbei
[6] http://zerofee.org/
[7] http://itunes.com/apps/senbei

And not only that, I've also made the source code available to anyone with a BSD license, as usual, in my Github account[8]!

Right now Senbei costs USD 0.99 (CHF 1.10), but I will increase the price until USD 4.99 (CHF 5.50) in every one of the next 5 updates. I have new features in the pipeline that I want to add... including an iPad version!



[9]

[8] http://github.com/akosma/Senbei/
[9] http://itunes.com/apps/senbei

Best Books of 2009

Adrian Kosmaczewski

2010-02-24

Every[1] year[2] I'm doing the same post (well, in 2006 I completely forgot to do it) that starts more or less with the same phrase: "every year I like to read at least 6 new tech books, and to learn a new programming language".

Last year's language was Go[3], and the books, well, here we go:

- Software Engineering: Barry W. Boehm's Lifetime Contributions to Software Development, Management, and Research[4]
- The iPhone Developer's Cookbook: Building Applications with the iPhone SDK[5]
- Beautiful Teams: Inspiring and Cautionary Tales from Veteran Team Leaders[6]
- Core Animation for Mac OS X and the iPhone: Creating Compelling Dynamic User Interfaces[7]
- Pragmatic Version Control Using Git[8]
- The 4-Hour Workweek: Escape 9-5, Live Anywhere, and Join the New Rich[9]

[1] /blog/best-books-of-2008/

[2] /blog/best-books-of-2007/

[3] /blog/thoughts-about-googles-go-programming-language/

[4] http://www.amazon.com/Software-Engineering-Contributions-Development-Practitioners/dp/047014873X%3FSubscriptionId%3D0F0YTN83N46JSX6KDT02%26tag%3Dakosmasoftwar-20%26linkCode%3Dxm2%26camp%3D2025%26creative%3D165953%26creativeASIN%3D047014873X

[5] http://www.amazon.com/iPhone-Developers-Cookbook-Building-Applications/dp/0321555457%3FSubscriptionId%3D0F0YTN83N46JSX6KDT02%26tag%3Dakosmasoftwar-20%26linkCode%3Dxm2%26camp%3D2025%26creative%3D165953%26creativeASIN%3D0321555457

[6] http://www.amazon.com/Beautiful-Teams-Inspiring-Cautionary-Veteran/dp/0596518021%3FSubscriptionId%3D0F0YTN83N46JSX6KDT02%26tag%3Dakosmasoftwar-20%26linkCode%3Dxm2%26camp%3D2025%26creative%3D165953%26creativeASIN%3D0596518021

[7] http://www.amazon.com/Core-Animation-Mac-iPhone-Programmers/dp/1934356107%3FSubscriptionId%3D0F0YTN83N46JSX6KDT02%26tag%3Dakosmasoftwar-20%26linkCode%3Dxm2%26camp%3D2025%26creative%3D165953%26creativeASIN%3D1934356107

[8] http://www.amazon.com/Pragmatic-Version-Control-Using-Starter/dp/1934356158%3FSubscriptionId%3D0F0YTN83N46JSX6KDT02%26tag%3Dakosmasoftwar-20%26linkCode%3Dxm2%26camp%3D2025%26creative%3D165953%26creativeASIN%3D1934356158

[9] http://www.amazon.com/4-Hour-Workweek-Escape-Live-Anywhere/dp/030735

Software Engineering: Barry W. Boehm's Lifetime Contributions to Software Development, Management, and Research[10]

Barry Boehm[11] is a name that might not strike a chord immediately, but if you work in the software field, it should. He has been working non-stop for the past 50 years (that's right, 50), discussing all kind of subjects related to the practice of software engineering. This book is a compilation of his most well-known papers, with subjects ranging from project management to components, from iterative techniques to developer productivity. The guy has written about all of it, and when you realize how right he was, you wish you had read those papers earlier in your career.

The iPhone Developer's Cookbook: Building Applications with the iPhone SDK[12]

Erica Sadun is a legend in the iPhone software engineering field. Her involvement with the iPhone developer community from the very beginning (during the dark times of jailbroken iPhones) has increased since the release of the official iPhone SDK in March 2008. Her articles on Ars Technica[13] or The Unofficial Apple Weblog[14] are epic, and her book could not be other than a masterpiece. Make no mistake: this is not a book for beginners (and, by the way, the second edition[15] has recently been published) but it is the perfect companion for all of us who spend a life in Xcode and the SDK. I hope she will continue providing more editions of this book, particularly now that the iPad has been announced, and will be released soon.

3133%3FSubscriptionId%3D0F0YTN83N46JSX6KDT02%26tag%3Dakosmasoftwar-20%26linkCode%3Dxm2%26camp%3D2025%26creative%3D165953%26creativeASIN%3D0307353133

[10] http://www.amazon.com/Software-Engineering-Contributions-Development-Practitioners/dp/047014873X%3FSubscriptionId%3D0F0YTN83N46JSX6KDT02%26tag%3Dakosmasoftwar-20%26linkCode%3Dxm2%26camp%3D2025%26creative%3D165953%26creativeASIN%3D047014873X

[11] http://en.wikipedia.org/wiki/Barry_Boehm

[12] http://www.amazon.com/iPhone-Developers-Cookbook-Building-Applications/dp/0321555457%3FSubscriptionId%3D0F0YTN83N46JSX6KDT02%26tag%3Dakosmasoftwar-20%26linkCode%3Dxm2%26camp%3D2025%26creative%3D165953%26creativeASIN%3D0321555457

[13] http://arstechnica.com/

[14] http://www.tuaw.com/

[15] http://www.amazon.com/iPhone-Developers-Cookbook-Building-Applications/dp/0321659570%3FSubscriptionId%3D0F0YTN83N46JSX6KDT02%26tag%3Dakosmasoftwar-20%26linkCode%3Dxm2%26camp%3D2025%26creative%3D165953%26creativeASIN%3D0321659570

Beautiful Teams: Inspiring and Cautionary Tales from Veteran Team Leaders[16]

O'Reilly has some very successful book series, like the "Head First"[17] and the "Beautiful" ones. The latter, very similar in spirit and nature to the "At Work"[18] series of books by Apress, provides a series of interviews to key industry players, in different fields, highlighting real-world experiences. This book takes this approach and brings an incredible series of war stories from organizations like IBM, Media Molecule or the NASA, told by Grady Booch, Tim O'Reilly, Cory Doctorow, Steve McConnell and, yes, even Barry Boehm. This book reinforced my belief that software is a social process[19], and I think that you will enjoy these stories about how many well-known products we use and love (or hate) every day have been brought to market, and how their teams struggled to stay together - or how they miserably failed.

Core Animation for Mac OS X and the iPhone: Creating Compelling Dynamic User Interfaces[20]

The iPhone OS and Mac OS X both share a legacy of design, attention to detail and awesomeness that can be explained by the sole existence of a single set of APIs: Core Animation[21]. This library allows developers to create stunning visual effects with great performance and with just a few lines of code. The rational use of animations is considered a huge usability win, bringing context awareness to users, helping them understand what's going on their applications and providing feedback and a "real world" feel to software. Bill Dudney[22] provides here a short yet complete introduction to the concepts behind Core Animation, both for the Mac OS X and iPhone OS; all in all a must have for all Cocoa and Cocoa Touch developers.

---

[16] http://www.amazon.com/Beautiful-Teams-Inspiring-Cautionary-Veteran/dp/0596518021%3FSubscriptionId%3D0F0YTN83N46JSX6KDT02%26tag%3Dakosmasoftwar-20%26linkCode%3Dxm2%26camp%3D2025%26creative%3D165953%26creativeASIN%3D0596518021

[17] http://oreilly.com/store/series/headfirst.csp

[18] http://apress.com/book/view/1430219483

[19] /blog/saving-a-failing-project/

[20] http://www.amazon.com/Core-Animation-Mac-iPhone-Programmers/dp/1934356107%3FSubscriptionId%3D0F0YTN83N46JSX6KDT02%26tag%3Dakosmasoftwar-20%26linkCode%3Dxm2%26camp%3D2025%26creative%3D165953%26creativeASIN%3D1934356107

[21] http://developer.apple.com/mac/library/documentation/cocoa/Conceptual/CoreAnimation_guide/index.html

[22] http://bill.dudney.net/

Pragmatic Version Control Using Git[23]

I've been a happy Subversion[24] user for years. I've kept svn repositories for my Master's degree work[25], my personal documents and of course for most of my projects. However, the server-centric nature of Subversion always made me think twice before creating a repository, and not being able to browse repository contents without a specialized client was always a pain in the neck. Not to name the fact that branching in svn is harder than it should be IMHO. Git[26] changed all of that. Creating repositories with Git is not only cheap, it's easy and fast, and branching could not be easier. This book was the one that showed me that there was a better way, and now with my Github account[27], I can't think of any other way to handle any kind of project. This book provided the initial knowledge to get started, and I strongly recommend it to anyone who wants to learn more about Git.

The 4-Hour Workweek: Escape 9-5, Live Anywhere, and Join the New Rich[28]

Tim Ferriss[29] is a strange kind of guy. He comes up with this book and tells you that you are working too much, that having a boss is killing you, and that you should be sipping margaritas in the Caribbean instead. And then he proceeds to show you how to do it. This book is interesting in many aspects, the first of which is the irreverent tone and the complete faith the guy has in his method. I could not agree with everything he said but I have to agree with the fact that he's really convincing. Tim believes in what he says and the book is a really funny one, and I can't deny that reading it helped me take the final decision to start my own company[30]. So, in any case, beware! This book is dangerous :)

[23] http://www.amazon.com/Pragmatic-Version-Control-Using-Starter/dp/1934356158%3FSubscriptionId%3D0F0YTN83N46JSX6KDT02%26tag%3Dakosmasoftwar-20%26linkCode%3Dxm2%26camp%3D2025%26creative%3D165953%26creativeASIN%3D1934356158

[24] http://subversion.tigris.org/

[25] http://remproject.org/

[26] http://git-scm.com/

[27] http://github.com/akosma/

[28] http://www.amazon.com/4-Hour-Workweek-Escape-Live-Anywhere/dp/0307353133%3FSubscriptionId%3D0F0YTN83N46JSX6KDT02%26tag%3Dakosmasoftwar-20%26linkCode%3Dxm2%26camp%3D2025%26creative%3D165953%26creativeASIN%3D0307353133

[29] http://www.fourhourworkweek.com/blog/

[30] http://akosma.com/

iPhone and iPad Developer Contest: win an iPad!

Adrian Kosmaczewski

2010-03-09

As a part of the DevDay for iPhone in London[1] and Geneva[2] you will have the chance to win one of three iPads!

These are the rules and conditions:

- It has to be a new application (for iPhone, iPod touch, iPad or, even better, all of them at once) not yet available in the Apple App Store;
- The deadline for registering for the contest is April 1st, 2010;
- The deadline for submitting your application is April 25th, 2010;
- Only one application per registered developer;
- The application will be tested by the Jury (composed by members of the program committee for the DevDays);
- The Jury will provide developers with the UDIDs of the evaluation devices.

[1] http://iphonedevday.com/iphone-london-2010/
[2] http://iphonedevday.com/iphone-geneva-2010/

The application will be judged using the following criteria:

- Originality;
- Usage of the mobile platform in general;
- Usability;
- Experienced quality;
- Visual design;
- Online and offline capabilities;

Do you want to participate?

- Submitted applications can be of any category, for example games, weather, news, finance, anything else!
- To register for the contest please send an email to contest@trifork.com[3]
- The winners will be announced at the conference in London and Geneva and at iphonedevday.com[4] just after the conferences.
- The three best applications will be rewarded with an iPad Wifi 16GB!

Dev Day for iPhone is a dedicated interactive one day conference for software developers and business professionals who want to learn how to successfully build and market iPhone applications. DevDay for iPhone will take place at Dexter House in London, UK, on April 26th, 2010, and in the Geneva Business Center in Geneva, Switzerland, on April 28th, 2010.

Key speakers at DevDay for iPhone include Raven Zachary[5] and Daniel Steinberg[6], experts in the field of iPhone application development and distribution. Raven Zachary, President of Small Society, who directed the Obama '08 iPhone application for the Obama Campaign will provide the keynote presentation. Daniel Steinberg is the editor for the new series of Mac Developer titles and the author of "Cocoa Programming".

In addition, the creators behind iPhone applications including Stanza, Deutsche Bank, Tag Heuer and Starbucks share their experiences of bringing iPhone applications to market.

[3] mailto:contest@trifork.com
[4] http://iphonedevday.com/
[5] http://iphonedevday.com/iphone-london-2010/speaker/Raven+Zachary
[6] http://iphonedevday.com/iphone-london-2010/speaker/Daniel+Steinberg

Suecia

Adrian Kosmaczewski

2010-03-14

en suecia, al menos en göteborg, hay luces en cada ventana.

dicho asi, parece una tremenda boludez, pero es asi: mirando las fachadas de cualquier edificio, cada ventana tiene un velador entre cada cortina. siempre. de esos veladores de mesa de luz, con su pantalla color crema, dando una luz acaramelada, melosa, calida, que contrasta con el frio exterior. en medio de la ventana, un velador, en cada ventana de cada edificio de cada avenida.

es una ciudad donde las calles estan cubiertas de piedritas.

dicho asi, parece otra tremenda boludez, pero ayuda a que la gente camine sin matarse entre los manchones de nieve, algunos a medio derretir y otros transformados en montañas de hielo y polvo. hay piedritas sueltas, que ayudan a que el zapato agarre mejor la calzada, a medio camino entre arena y canto rodado, en cada tramo de cada vereda de cada avenida.

es un pais raro.

los restoranes estan repletos a las 15 como en españa, pero vacios a las 19 como ni siquiera en suiza. la gente desayuna panceta con huevo y porotos con tomate, pero no hay gordos en las calles; es mas, son todos flacos de un metro noventa promedio. son tan rubios que a los albinos les dicen morochos. en los tranvias, las maquinas que te venden el boleto tienen un boton que dice “english” pero que igual te da las instrucciones en sueco. la gente es seca pero cordial, parece que te van a mandar a la mierda en cualquier momento; y cuando te ven con un mapa se paran y te preguntan si necesitas ayuda, con una gentileza que desmorona. pronuncian las “a” como “o”, y la “y” suena como una “u” francesa. y si es una “í¥” con redondelito es diferente de si es una “ä” con dieresis. hay locales de venta de “gudis” por todos lados, vendiendo golosinas a granel; agarrate una bolsa en la entrada y paga a la salida, al peso.

deci que no manejan por la izquierda, como los ingleses, eso ya seria mucho.

Color Sin Dolor

Adrian Kosmaczewski

2010-03-24

viejita,

te fuiste durante la mañana del 15 de marzo del 2010.

seguramente la noche anterior cenaste, miraste la television, tomaste tus medicamentos, despues te acostaste mirando la foto de tu hermano charles, y te quedaste dormida.

te quedaste dormida, mama, sin dolor.

creo que fue la mejor muerte que a uno le puede tocar. y cuando miro para atras, y veo el kilometraje que tenias, creo que fue la mejor. mucho dolor hubo, mucho.

tu cara tenia paz, no se como describirlo. cuando te vimos estabas tranquila, tenias incluso ese guiño en el ojo izquierdo que tenias cuando dormias. te fuiste soñando, probablemente con tu hermano charles, con tu perra brigitte, tal vez la abuela herta estaba ahi. quizas tu viejo, buscando un perdon que solo alla arriba se puede obtener. por ahi estara d'agosto, rini, walter, el tata kazimierz tambien. incluso la negra sosa, que canto en tu honor durante la ceremonia.

claudia te maquillo, y yo puse entre tus manos un crucifijo. tu fe te ayudo a dejar este mundo en paz. tambien puse entre tus manos cartitas y dibujos que te hice cuando era nene. todo eso se fue con vos. tenias tu pulover preferido, el negro y rojo, te quedaba bien.

aca quedo el dolor. como siempre, los que la pasamos peor en estas situaciones son los que nos quedamos. el viejo estaba destrozado. el te queria mucho, a pesar de los años y de la distancia.

rezamos mucho, y no solo aqui en suiza. hubo rezos para vos en argentina, bolivia, inglaterra, uruguay, incluso en suecia. la gente se movilizo. vos ya lo sabes, quizas; el mundo es un poco mas triste desde que te fuiste. pero si la oracion sirve, entonces quiere decir que no te fuiste sola, sino acompañada.

yo, al menos, veo un mundo con otro color, con otro dolor. no se como explicarlo, pero las cosas tienen otro sabor, otra textura. no es mas gris, no es mas colorido, es simplemente diferente. paradojicamente, yo tambien estoy mas en paz.

no fui un hijo perfecto, vieja, pero siempre quise que seas feliz.

es lo que te susurre al oido la ultima vez que te vimos, antes de la ceremonia, te dije que seas feliz. donde quiera que estes, quiero que dejes esa mochila de dolor y la reemplaces por una de color.

hiciste mucho para que yo sea quien soy hoy. ser hijo unico tiene un peso y una importancia particular, que determina muchas cosas en una personalidad. en el 96 yo quise ser yo mismo, y me escape de tu vida, lo cual, yo se, te dejo desamparada. despues, nunca mas fui el adri de la infancia, porque creci de golpe y a los cachetazos.

hice lo posible para no defraudarte, para ser alguien de bien.

encontre en la biblia la notita que me escribiste, supongo yo, alla por los ’80, y la lei en publico durante la ceremonia del jueves pasado. me dejaste un mensaje hermoso, que ire entendiendo poco a poco.

porque tuve que hacerme adulto para entenderte, algo que no se podia hacer mientras yo era nene. lo intente, pero eso me costo mi infancia. ahora estoy tratando de recuperar esa infancia perdida, con clau. ella tambien tuvo que crecer de golpe.

vos tambien, tuviste que crecer incluso mas rapidamente que nosotros. te robaron tu infancia y eso marco tu sensibilidad, te robo sueños y te dejo desamparada. tengo mucho odio por todo eso, no te lo merecias. eras buena persona.

a pesar de todo, sabiendo tu historia, no puedo sino admirar lo tuyo. porque despues de tu hospitalizacion del 2008, “te cayo una ficha,” y lograste encontrar el perdon y nos dimos cuenta todos de eso. estabas en paz con vos misma, estabas mas tranquila. el histerico era yo, y te pido perdon por este caracter volcanico mio.

pero vos lograste estar en paz. viniste a casa a pasar navidad, y fue una noche hermosa y fantastica. hicimos un ultimo viaje a schaffhausen, las viste a milena, renata, daniela y hella. fuiste cerrando capitulos, poco a poco, como un escritor que va terminando su libro.

vamos a llevar tus cenizas a argentina. siempre tuviste nostalgia de buenos aires. yo tambien. “deshauciado esta el que tiene que marchar / a vivir una cultura diferente.”

me viste en casa, con mi mujer, con trabajo, salud, e incluso con un titulo universitario. creo que viste que ya habia logrado cierto equilibrio, cosa rara en mi. eso te preocupaba mucho. creo que te tranquilizo ver que estoy mejor que nunca.

y te fuiste en paz, espero, dejando ese dolor, cambiandolo por color. yo tambien bese tu frente, por ultima vez, esperando que tu amanecer sea radiante, repleto de colores, como un arco iris.

te quiero mucho, vieja.

Why Some Analysts Don’t Get The iPad (Yet)

Adrian Kosmaczewski

2010-03-26

A couple of days ago I’ve forwarded this article on The Apple Blog[1] to a friend of mine: “Why Apple’s iPad Can’t Succeed in Schools (Yet)”, by Liam Cassidy[2].

This article, even if written in a supposedly “pro-Apple” blog, is rather critic and makes some debatable assertions at best, some of them without any proper backing in terms of analysis or statistics.

But most importantly, this article shows that the iPad can really become the disruptive device we’ve been waiting for years. The expectations are huge. The inflection point of IT in education could be April 3rd, when the iPad will start its career in the hands of users.

My friend (who works in the education field, by the way) answered back to me, and boy, did he make a point. Judge by yourselves:

[1] http://theappleblog.com/

[2] http://theappleblog.com/2010/03/24/why-apples-ipad-cant-succeed-in-schools-yet/

> Apple has started making the iPad available on its online education store in packs of 10 with an appallingly–stingy discount of only $20 per iPad. If Apple wants to start a computing revolution with the iPad, it absolutely must get the device into schools.

Why must?

> But in order to do that, it's going to have to try a lot harder, and generous discounts are the easiest problem to solve.

Low price (lowest price Apple "computer" I've ever seen!) is not enough? Generous discounts will change the penetration?

> There are much bigger hurdles standing in the way.
>
> Let's start with costs alone. Assume a school wants to buy an iPad for each of its students. Assume the school is small with only 300 children enrolled. Assume also that the school wants to buy the cheapest iPad without AppleCare. At a little more than $450 per iPad, that's a cost of almost $144,000. I imagine the average state-funded school enjoys less than half that in its annual I.T. budget.

Yes, and? What is the school's annual book budget? Can you imagine that amount too?

> "Aha!" you might argue, "Many schools in underprivileged areas get subsidies from the state and provide laptops for their pupils."
>
> And, of course, you'd be right. Many schools do provide their students with free or 'nearly-free' laptops. But not decent laptops. We're talking cheap, disposable netbooks that cost far less to insure against loss or damage. (Let's be realistic – the younger the student, the greater the chance of laptop-death!)

Weird. So schools buy cheap. Schools buys cheap furniture? Cheap books? Really?

> I graduated from High School back in the early 90s, and even then my school was considered **ahead of the curve** when it came to the adoption of computer technology in class. Even so, there were no Macs in my school. They were just too expensive. Here in the UK, the fierce battle in the 1980's between Acorn, Sinclair, Atari, Amstrad and Commodore meant that there were many perfectly capable, cheap microcomputers available to schools. The Mac was superior to those machines in almost every way, but it couldn't compete on price.

(emphasis added)

I wonder how much money schools spent on computers then. Did an Acorn, Sinclair, etc. cost the equivalent of today's $499? Different

times, obviously.

> It has been 16 years since I graduated from high school. And while I'm happy to report that my old school now has iMacs in most classrooms, sadly they only run Windows XP.

Why did they buy iMacs then? They were the cheapest?

> The reason for this comes down to two simple factors; Cost, and What's Best for the Kids. It seems more educational titles are available at lower prices on Windows than on Mac OS X.

Define "Educational Title". Seriously: define it, today, as I suspect the market has slightly changed since Grandma and Me CD-ROMs[3]. How about counting how many Living Stories[4] run on the PC platform?

> And, outside school, the kids encounter more Windows PCs than Macs.

And? Since when does school teach to use a platform? They educate pupils on what to do properly with Computer in this Digital age, they don't (as far as I know) deliver Windows certificates. At least they should not.

> So I look at the upcoming iPad and, even though I can see the potential it offers to schoolchildren (and the wider education market),

… not sure you do, actually…

> I can't help but wonder if it has any real chance of making a dent at this time. HP's upcoming slate PC has more chance of being adopted by my old school simply because it works with all their existing software and runs Windows — the platform the school believes the pupils are better served knowing,

You believe this, not sure the school actually does.

> rather than Mac OS X, which they have concluded is just too obscure and "specialist."

Weird argument for an "ahead of the curve" school (?).

> And as though these fiduciary and policy-driven decisions aren't bad enough, there's another glaring challenge to getting the iPad widely accepted in schools; at the end of the day, it's just not a book.

You're right, it's much less…

[3] http://www.taglearning.com/productdetails/Just+Grandma+and++Me+:+Living+Books+Series.html

[4] http://itunes.apple.com/ch/app/my-living-stories-little-red/id363506793?mt=8

> You see, tablets-as-books is a great idea until the battery dies, and then the student has no textbook and no computer. She will have to plug-in to a power outlet if she wants either of those things back.

A school with iPad will make sure the pupils understand they must have a fully charged iPad when they arrive at school. Anyway kids hate having to plug something in during work, they'll do this at home.

> But consider the delicate health and safety issues associated with cable-safety in a classroom environment.

Delicate health and safety issues? Tripping on a cord is a delicate issue?

> Not to mention the maintenance costs (that's a lot of power outlets being used more than ever before)

Maintenance is not the usual budget line where you put electricity bills.

You're right, that a lot. I've done the calculation for a Mac mini, dissipating 13W when used. If a school uses it 25 hours a week, 40 weeks a year, for 5 years, it's 5000 hours of use. With a kWh priced at 20 centimes, that's an electrical cost of 13 swiss francs. Thirteen. In five years. And the iPad is going to draw much less power (and be used maybe more hours a week, probably).

> and don't forget the school will suddenly incur higher energy bills. Say what you will about a paper-textbook, at least it doesn't need plugging-in.

You sometimes need light to read. Oh, sorry, a candle will do.

> And then there's the issue of damage. What happens if an iPad screen is cracked? A damaged book cover doesn't render the book's contents inaccessible, nor is it likely to slice into fingers. Plus, the cost of a replacement book is trivial. Remind me how much the cheapest iPad is?

I see: compare one book to one iPad. Hm.

A school having a 1-1 project with MacBooks did an insurance system, every kid paid a little sum in a fund. In case of loss or damage, the kid had only 30% of the device to pay back. They saw very little damages, kids took good care of the laptops. The more precious the device, the more care they took, actually.

> Oh, and let's not forgot that Apple isn't perfect. Remember when the iPhone OS was updated to 3.1 in September last year? I wrote about it here[5], and the comments quickly ran to over 100. iPhones everywhere were freezing, crashing,

[5] http://theappleblog.com/2009/09/14/iphone-os-3-1-update-causing-crashes-on-iphone-3gs/

> and generally just refusing to work, and all as a result of an official update from Apple itself!

Since when do 300 kids perfom the update on the first day Apple releases it? I bet that kids and teachers would update at best their iPad once a year, and probably not before someone from IT gives it a go.

> What happens when Apple does the same thing with the iPad? Even the most diligent students who take the greatest of care with their always-charged-in-time-for-class iPads will suffer if an update from Apple proves flaky.

Hell! That's a serious risk!

> And, finally, there's the matter of crime. No one ever wanted to rob a kid from my school. The only thing we ever carried in our bags was biology books and the occasional Thundercats pencil case. But what if my school handed-out iPads to its pupils? Overnight, the school uniform would become an advertisement to any would-be criminal; "mug this kid – expensive computer on-board."

iPad = Expensive computer... oh dear... Kids don't have to take the iPad home, depending on the age of the kid, the school will keep them safe.

> I'd dearly love to see all school kids and college students everywhere take-up iPads as their favorite learning tools. Sadly, I just don't see how that can happen as long as they remain significantly more expensive than textbooks, more sophisticated than simple e-book readers and less resilient than the existing, proven toolset — traditional, dead-tree textbooks.

You don't see? Watch.

akosma software is hiring

Adrian Kosmaczewski

2010-03-29

Due to unprecedented growth, **akosma software is hiring**.

Here's what we offer:

- A full-time position as a **partner software developer** (for the moment, **just one**, but stay tuned). Your primary task will be to create and maintain software products, either our own or our clients'.
- But beware: coding is only part of the job: you are also going to write documentation, evaluate projects, manage schedules, update websites, train other people, answer e-mail, update Twitter accounts, perform a few sysadmin tasks, and actively create your own work environment.
- You are also going to blog and to publish open source projects. A lot. Your skills will be seen, published, under your own name.
- Telecommuting: work at home if you want. No fixed work hours. As long as the work is done, we don't mind. Skype[1], Redmine[2] and iChat[3] are your allies.
- A workplace with a Joel Test[4] of 10 (we still don't have a hallway to do usability testing, and you will be the first "new candidate")
- A good salary (by Swiss standards, which tend to be pretty good compared to other countries)
- All the nice things of a small company (and some drawbacks too, why deny it, that's how life is.)

You have the following characteristics and skills:

- Excellent, brilliant iPhone development skills: you breath Objective-C, Xcode, Interface Builder, and everything else bundled in the SDK. You downloaded the first iPhone SDK two years ago and immediately fell in love with it. You can write a Twitter client in half a day of work.

[1] http://www.skype.com/
[2] http://www.redmine.org/
[3] http://www.apple.com/macosx/what-is-macosx/ichat.html
[4] http://www.joelonsoftware.com/articles/fog0000000043.html

- At least two (2) applications published on the App Store (in any country), even if under the name of a previous client (in this case the reference must be verifiable).
- Strong online presence: an interesting blog, a popular Twitter[5] account, a nice StackOverflow[6] profile, an awesome Github[7] or bitbucket[8] account with some cool projects, other contributions to open source, all are good.
- Excellent knowledge of C++ and Ruby.
- You know how to use a basic image editor like Pixelmator, Acorn or even Photoshop. You know that PNG rocks. You have read the Mobile HIG[9]. You bought a developer sketchbook[10] and you keep it with you at all times.
- Good knowledge of web development, that means that you've used one or many of the following: Rails[11], jQuery[12], Prototype[13], PHP[14], JavaScript[15], Cappuccino[16], Sproutcore[17], WebObjects[18], or other similar libraries, technologies, and buzzwords.
- Self driven, impulsive, opinionated and creative. You are a doer, an actor, and it shows. People either love or hate you.
- You use Git[19] or Mercurial[20] every day. And you always write a meaningful message when you commit.
- Whatever your native language is, you have excellent English skills, written and spoken. Particularly, you know the difference between "it's" and "its".
- Unit testing (well, quality management in general) is a reflex, not an afterthought.
- You feel more confortable and work faster using a command line. You donwload files with curl[21], you connect to your home server with ssh[22] and you edit your blog using vim[23]. Please include your .tcshrc, .bashrc and .vimrc files together with your CV.

---

[5] http://twitter.com/

[6] http://stackoverflow.com/

[7] https://github.com/

[8] http://bitbucket.org/

[9] http://developer.apple.com/iphone/library/documentation/UserExperience/Conceptual/MobileHIG/Introduction/Introduction.html

[10] /blog/roundup-of-iphone-app-sketchbooks/

[11] http://rubyonrails.org/

[12] http://jquery.com/

[13] http://www.prototypejs.org/

[14] http://php.net/index.php

[15] http://javascript.crockford.com/popular.html

[16] http://cappuccino.org/

[17] http://www.sproutcore.com/

[18] http://en.wikipedia.org/wiki/WebObjects

[19] http://git-scm.com/

[20] http://mercurial.selenic.com/

[21] http://curl.haxx.se/

[22] http://en.wikipedia.org/wiki/Secure_Shell

[23] http://www.vim.org/

- Strong Mac OS X skills. You prefer SubEthaEdit[24], TextMate[25] or Smultron[26] when you don't use vim (see previous point).
- Minimum age: 30 years old, with a proven minimum of 5 years of software engineering experience in general (not only iPhone, of course). We care more about practical experience than university degrees, but hey, if you have one, even better!

Bonuses (not required but a good way to score more points):

- You are located in Switzerland. Yup, depending on your skills, we could work out a solution for you even if you live abroad.
- Experience in development of software for "jailbroken" iPhones (of course, if you have apps published in Cydia, that's a bonus++).
- Experience in the development of iPhone apps using other frameworks[27] than Apple's own SDK.
- Experience in the development of Android, BlackBerry, or other (ir)relevant mobile platforms.
- OpenGL, audio, signal treatment, any kind of multimedia programming or even graphic design skills.
- You write letters to your beloved ones in LaTeX[28].
- You are a musician, a painter, a designer, an artist. Many artists are also great software developers.
- Your blog uses a weird combination of cutting-edge CSS3 selectors, obscure page generators and some innovative JavaScript libraries.
- You own a Das Keyboard[29]. Preferably the one without labels on the keys.
- You have read Getting Real[30] and It's Not How Good You Are, Its How Good You Want to Be[31] and tried desperately to apply that knowledge in a previous workplace with abysmal Joel Test[32] scores. You didn't succeed and that's why this blog post sounds interesting.
- You have experience in other SCM systems, proprietary or not, like Subversion, Bazaar or even CVS. SourceSafe does not add points: actually it substracts, so you'd better not mention it.
- You know other not-so-common programming languages, like Haskell, Erlang, Brainfuck, Lisp or Prolog. Java has the same effect as SourceSafe – see previous point (unless you have done

[24] http://www.codingmonkeys.de/subethaedit/
[25] http://macromates.com/
[26] http://www.apple.com/downloads/macosx/development_tools/smultron.html
[27] /blog/iphone-apps-without-objective-c/
[28] http://www.latex-project.org/
[29] http://www.daskeyboard.com/
[30] http://gettingreal.37signals.com/
[31] http://www.amazon.com/Its-Not-How-Good-Want/dp/0714843377
[32] http://www.joelonsoftware.com/articles/fog0000000043.html

Android programming, in which case you are forgiven, of course).
- A Bachelor, a minor, a major, postgraduate, Master's or PhD degree in anything else but computer science. In other words, show us that you have a life and that technology is just another one of your passions.
- Your new year resolutions for 2010 include reading at least 6 technical books and learning a new programming language.
- You are convinced that compilers are just a poor man's testing suite.
- You speak or write other "human" languages, like Chinese, Arabic, Quechua or Swahili (but please, only contact us in English.)
- You shake your head in dismay while reading this posting and the sheer amount of bad geek jokes.
- Finally, you know Linux and Windows pretty well. As a developer, that is, not only as a user. C++, .NET, Python, all are welcome in this sense. But not Visual Basic, in any of its forms.

Are you interested? Then get in touch[33] presenting yourself (in English please), and explaining in detail why you think you are the best candidate we have ever met and how lucky we are. Don't forget to add the link to your LinkedIn account or your CV (PDF, Pages, Word, OpenOffice, LaTeX or plain text accepted).

By the way, the presentation letter is very important.[34] Don't skip that part. We won't pay as much attention without a good one. We'll get back to you anyway, even if your skills aren't what we are looking for; we hate those companies who ask for workforce, receive candidacies and never answer them back to say "no thanks", so we don't behave that way.

[33] /about/

[34] http://37signals.com/svn/posts/1748-forget-the-resume-kill-on-the-cover-letter

iPhone Apps Development in Berne @ the University

Adrian Kosmaczewski

2010-03-30

Apple Switzerland just sent the following e-mail, which, I was informed, can be publicly disclosed. Steven Woolcock[1] (Apple Worldwide Developer Relations) will be in Switzerland that day; this unique event cannot be missed in any way!

> **iPhone Apps Development in Berne @ the University**
> Date: Friday, 16.4.2010 Time: 9am – 3.30pm
>
> Dear customers Dear partners
>
> We are pleased to invite you for an exciting iPhone event on the 16th of April 2010 in Berne. This seminar is dedicated to the development of applications running on Apple's mobile devices (iPhone, iPod Touch, iPad). Emphasis will be given to learning applications developed by educational institutions and students. The seminar will include an overview of the development tools and techniques that allow the building of native as well as web-based applications followed by demos.
>
> We will discuss what makes great mobile applications and how educational institutions can define new paradigms in mobile application development. Faculty members from european universities will demonstrate how universities are teaching iPhone application development to their students, and show what students are able to deliver when they are working for real projects from within the local industry.
>
> Finally we will introduce the IT Innovation award of the Swiss Milton Ray Hartmann Foundation. Valued around CHF 30.000, the "Goldene Maus/Souris d'Or 2010" will be awarded to the developer(s) of the most innovative mobile app at Worlddidac in Basel in October 2010.
>
> Attendees should be interested in:
>
> - how to develop a successful app
> - how to gain CHF 30,000 for the most innovative app

[1] http://uk.linkedin.com/in/woolcock

- meeting the community

Registration links and more details: The event is open to registered guests only!

CHde: http://seminars.apple.com/goToEvent.html?id=96210&locs=ch_de
CHfr: http://seminars.apple.com/goToEvent.html?id=96210&locs=ch_fr

We are look forward to seeing you or any other interested people of your entourage in Berne.

Kind regards, Education Team Apple Switzerland

Application iPhone ThierryWeber.com

Adrian Kosmaczewski

2010-03-31

Vous aussi vous hésitez ? Vous vous posez des questions du genre “devrais-je m’y mettre ?” “Devrais-je essayer ?” Cette vidéo est là pour enlever tous doutes et répondre à vos questions. N’attendez plus, lancez-vous ! ThierryWeber.com[1] se fera un plaisir de vous aider :) A bientôt si c’est pas avant sur l’App Store[2].

[3]

[1] http://www.thierryweber.com/
[2] http://itunes.apple.com/ch/app/thierryweber-com/id342010688?mt=8
[3] http://www.thierryweber.com/

CulturePod.ch: Interview of Adrian Kosmaczewski by Thierry Weber

Adrian Kosmaczewski

2010-04-01

Adrian Kosmaczewski, the CEO of akosma software talks[1] to the microphone of CulturePod.ch[2]. Whether the importance of ensuring the communication via a device like the iPhone or iPad but also the list of applications, Adrian explains the opportunities of this platform.

[3]

With his catalog of applications such as SwissAlert[4], Notitas[5] or Bluewoki[6], he is mostly behind the development of the official iPhone application[7] of Lift, the famous swiss conference[8] (Life Idea, Future and Technology) launched by Laurent Haug. An application that has emerged as a result of collaboration between akosma software and ThierryWeber.com[9]. In addition to the cap developer he's also organizing conferences about mobile market[10] and especially the iPhone. Adrian is also writing a book on the same subject.

[1] https://www.youtube.com/watch?v=hTq7Y0bLz2w
[2] http://culturepod.ch/
[3] http://www.culturepod.ch/
[4] http://itunes.com/apps/swissalert
[5] http://muchasnotitas.com/
[6] http://bluewoki.com/
[7] http://itunes.com/apps/liftconference
[8] http://liftconference.com/
[9] http://www.thierryweber.com/
[10] http://devdayforiphone.com/iphone-geneva-2010

First iPads in Switzerland!

Adrian Kosmaczewski

2010-04-07

Queuing in front of the Fifth Avenue Apple Store[1], 767 Fifth Avenue, New York City, Saturday at 0515:

Getting into the Apple Store at 0900:

The most important part: buying the iPads! This photo was taken by Associated Press and later relayed by Le Figaro[2] (6th photo) and Tages Anzeiger[3] (5th photo), among others:

Arriving to Switzerland[4], as filmed by Thierry Weber[5]:

Talking about the iPads in yesterday's geek night in Zurich, with Jonas Schnelli from include7[6] (photo taken by Jørn Larsen from Trifork GmbH[7]):

[1] http://www.apple.com/retail/fifthavenue/

[2] https://web.archive.org/web/20100408024050/http://www.lefigaro.fr/sciences-technologies/2010/04/03/01030-20100403DIMWWW00482-lipad-debarque-aux-etats-unis.php

[3] https://web.archive.org/web/20100406082444/http://www.tagesanzeiger.ch/digital/computer/Die-iPadra-hat-begonnen/story/10568509

[4] https://www.youtube.com/watch?v=aehFyov3mjA

[5] http://www.youtube.com/watch?v=aehFyov3mjA

[6] https://web.archive.org/web/20100506002120/http://www.include7.ch/

[7] http://trifork.ch/

akosma software was there, in the grand opening day, to get a couple of iPads before its general availability in Switzerland :) Contact us![8] We can and we want to help you bring your ideas to the iPhone and the iPad.

[8] /about/

iPad for Developers Presentation

Adrian Kosmaczewski

2010-04-07

This is the presentation about the new features in iPhone OS 3.2 and the iPad in yesterday's geek night, at Zurich's Technopark, done together by Jonas Schnelli from include7[1], Jørn Larsen from Trifork GmbH[2] and akosma software.

iPad For Developers[3]

View more presentations[4] from Adrian Kosmaczewski[5].

Thanks to everyone (over 80 people) who attended! Some photos of the event, courtesy of Jørn Larsen[6]:

[1] http://include7.ch/
[2] http://trifork.ch/
[3] http://www.slideshare.net/akosma/ipad-for-developers
[4] http://www.slideshare.net/
[5] http://www.slideshare.net/akosma
[6] http://twitter.com/jlatrifork

Figure 1: z3t.jpg

Figure 2: ukk.jpg

Interview by iPhoneDevSuisse

Adrian Kosmaczewski

2010-04-09

Last Wednesday I was interviewed by Junior[1] from the iPhone and iPad Dev Group website[2] (@iPhoneDevSuisse[3] in Twitter).

We talked about the newly-released iPad and the possible business cases around it, about the rumored features of iPhone OS 4.0 (remember, this was on Wednesday) and also a bit about akosma software and our new developments.

Given that Junior is a Brazilian-Italian, and that I'm from Argentina, this conversation (in English) might really sound funnier and contain more grammar mistakes than expected :)

Thanks Junior! I welcome his initiative and I urge you to visit his website[4] and follow him on Twitter[5] (part 1[6] and part 2[7].).

[1] http://www.bonto.ch/
[2] http://www.iphonedevgroup.ch/
[3] http://twitter.com/iPhoneDevSuisse
[4] http://www.iphonedevgroup.ch/
[5] http://twitter.com/iPhoneDevSuisse
[6] https://www.youtube.com/watch?v=1HkxZ2dr6zI
[7] https://www.youtube.com/watch?v=M1CTrYjYJ40

Mobile Advertising, by Edipresse

Adrian Kosmaczewski

2010-04-13

Yesterday I attended this month's Mobile Monday meeting in Bern[1], and among all the presentations, the best ones (in my opinion, they were outstanding) were those of Nicholas Heller from Google and Marc Lamarche[2] from Edipresse. I will quickly summarize this last one in this blog post.

[3]

Marc did a great job of presenting Edipresse[4] to an audience coming from all over Switzerland, striking a chord when he mentioned that recently Edipresse merged[5] with tamedia[6], which is the owner of major German-speaking newspapers such as the SonntagsZeitung[7], 20 minutes[8] or the Tages Anzeiger[9]. Edipresse enjoys a nearly monopolistic position in the French-speaking part of Switzerland, as a matter of fact.

Statistics

Marc's presentation consisted of very interesting statistics of mobile traffic in some of Edipresse's mobile apps and websites. For example, did you know that LeMatin.ch[10] mobile website drives more than 15% of the total traffic generated by all of Edipresse's sites?

[1] http://www.mobilemonday.ch/?page_id=2115
[2] http://twitter.com/MarcLamarche
[3] http://www.mobilemonday.ch/
[4] http://www.edipresse.com/
[5] http://www.newspaperinnovation.com/index.php/2009/09/17/edipresse-tamedia-merger-approved/
[6] http://www.tamedia.ch/
[7] http://www.sonntagszeitung.ch/
[8] http://www.20min.ch/
[9] http://www.tagesanzeiger.ch/
[10] http://www.lematin.ch/

One of the most striking statements of Marc's presentation was that 86% of the visits to Edipresse's websites comes from iPhone OS devices! This, in turn, means that currently most of Edipresse's online mobile advertising efforts are directed towards this platform. I think that the outcome of the iPad will help maintain this level, while the Android platform grows at the same time.

Other interesting stats showed that mobile traffic peaks at 8am and 8pm, while during "office hours", pure web traffic is higher; the conclusion is that most of mobile traffic is done when people travel from or to their jobs, and from home, in the evenings.

Amusingly, in the graphics there was a clear indication of a new trend towards "in the bed browsing" late at night, which somehow confirms my theory that we are entering the era of Couch Computing[11]. Of course, the iPad will only strengthen this phenomenon.

Web apps vs. native apps

In a relatively debatable note, Marc's assertion that web apps will take the lead in the next 10 years left me rather unimpressed, as we've heard similar claims in the past 10 years, precisely, but this never happened (at least until now). Marc's argument has to do with evident cross-platform issues. Clearly understandable, since supporting all of these platforms with the same content requires added costs for Edipresse, while this problem is already solved in the web, at with the price of a somewhat less pleasing user experience.

I think, in any case, that "native" platforms still have a bright future ahead, as do web apps. Both have their relative strengths[12] and can complement each other perfectly well. Particularly, the raise of "App Stores" for most mobile platforms provides an increased level of visibility for companies looking to expand their brands, which is a differentiating added value that web apps do not provide (yet).

Formats

Edipresse provides advertising in, basically, two formats:

- Full screen
- Banners

Currently they are also testing contextual, geo-localized advertising. I'm interested in knowing how it works for them, particularly now that there are rumors[13] that since the release of the iAds platform by Apple, the new clause 3.3.9 in the developer agreement restricts the use of device data information for advertising.

[11] /blog/the-dawn-of-couch-computing/
[12] /blog/webtuesday-mobile-app-frameworks/
[13] http://www.148apps.com/news/apple-create-mobile-ad-monopoly-iphone/

Full-screen format

Full screen advertising is pre-downloaded and shown on next run of the application, but only for 3 seconds, and is saved for 2 days. This format is both “clickable” and “skippable”, and in the case of the iPhone OS, it is designed to make the user stay in the application, by embedding the target website in the app.

Interactivity

Edipresse is also exploring different interactivity options with mobile ads; so far, these are some of the most interesting strategies used worldwide:

- SMS (very popular in third world countries, but less popular in Europe)
- “Quick Response” or QR code scanning, with codes in street advertising, in newspapers or even bus stops. Good business results! (particularly in Japan)
- Image recognition

Marc mentioned also some of the advantages of interactive printed ads with a mobile twist:

- Tracking and statistics
- Interactivity
- Direct marketing

Conclusion

All in all, I strongly recommend meetings like this; they are far enough from my day-to-day duties to be an interesting source of new knowledge, without being completely alien at the same time. The networking session afterwards was also a good moment for sharing a drink with interesting folks and getting to know better the current mobile landscape in both sides of the Sarine[14]!

PS: just as a side note, this blog post was written as a Pages[15] document on my iPad, expanding the notes taken during Marc’s presentation. The capabilities of this small device for note taking and for a quick writing during the trip back home are nothing short of extraordinary.

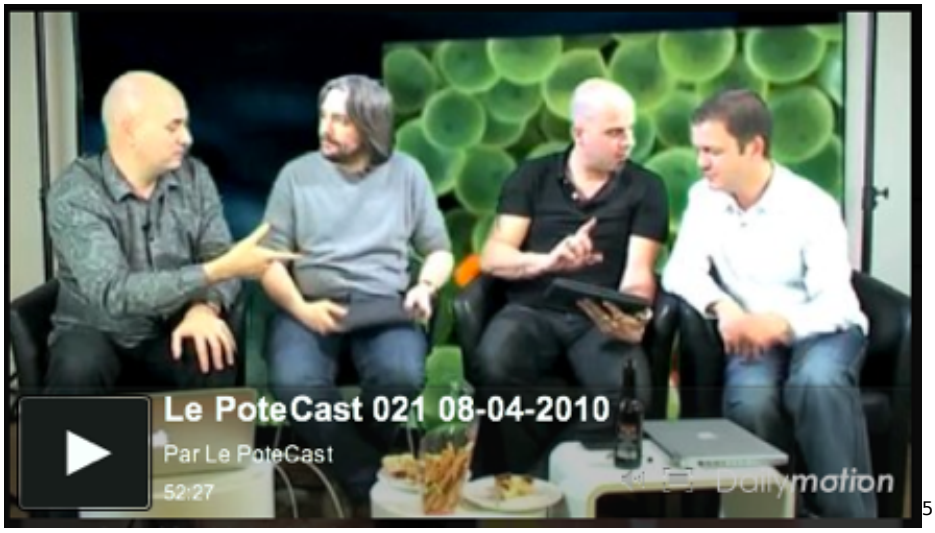
[14] http://en.wikipedia.org/wiki/Saane/Sarine
[15] http://www.apple.com/ipad/features/pages.html

Le Potecast 21

Adrian Kosmaczewski

2010-04-14

La semaine dernière j'ai eu l'immense plaisir de partager un moment de fun avec l'équipe déjantée du Potecast[1]! Merci a Thierry Weber[2], Jay-Rôme Berthoud[3] et Nicolas Pittet[4] pour leur accueil, et pour la discussion super intéressante autour de l'iPad, de la toute nouvelle version de l'iPhone OS 4.0 et bien plus encore.



[5]

A voir sur le site du Potecast[6] ou sur Dailymotion[7].

[1] http://www.lepotecast.com/
[2] http://www.thierryweber.com/fr/
[3] http://www.chicheux.ch/
[4] http://www.inthebox.ch/
[5] http://www.dailymotion.com/video/xcwgin_le-potecast-021-08-04-2010_tech
[6] http://www.lepotecast.com/site/2010/04/12/le-potecast-21/
[7] http://www.dailymotion.com/video/xcwgin_le-potecast-021-08-04-2010_tech

Making Traveling Enjoyable Again

Adrian Kosmaczewski

2010-04-20

If there's only one good thing we could take from the global grounding of planes all over Europe, it might as well be the possibility to enjoy traveling again. Even recognizing that the airline industry has been able to dramatically cut costs and times of travel, one can't deny the fact that it has done nothing to increase the pleasure of traveling. Quite the opposite, as a matter of fact.

To put it elegantly, traveling by plane is a pain in the neck. In the 90's it wasn't better, but at least the Twin Towers were still standing in their place and there wasn't a new "terrorist threat" every year or so, making the life of the rest of the travelers an ongoing misery.

Taking a plane exposes you to a staggering amount of things that can go wrong, from the most complex to the most ridiculous. They keep on telling us that traveling is the most secure way to travel, but they say nothing about the ever smaller and more uncomfortable seats, about the shitty food they keep on serving and the increasing number of destinations they keep on sending our luggage, more often than not exactly the opposite one we are going to. Without mentioning the amount of cancelled flights without warning, the non-guaranteed connections, the unbelievably ridiculous schemes of ticket pricing (why a return ticket is cheaper than a one-way is beyond me) and the oh so many other things that make air travel an utterly miserable experience.

Oh, but it is the most secure way of traveling. Yeah, right.

Disclaimer: I've been a Swissair[1] employee in the 90's, so I know a bit of how an airline can go every year a bit worse, until it disappears completely from the face of Earth.

So now the ashes of Iceland have grounded the planes of a whole continent, generating losses of around 300 million dollars per day[2]. Ups (and, by the way, what an irony and a colorful way Iceland has found to return Britain the favor of an incredible economic disaster, of which it was the biggest victim[3] but not the most important contributor… I

[1] http://en.wikipedia.org/wiki/Swissair

[2] http://www.businessweek.com/news/2010-04-19/european-carriers-seek-aid-for-ash-losses-british-airways-says.html

[3] http://en.wikipedia.org/wiki/2008â€“2010_Icelandic_financial_crisis

do think nature has a sense of irony after all.)

What I find interesting is that, if the ashes keep on clearing the sky from those shitty winged artifacts filled with unhappy travelers, we will have a chance to slow down, and we might as well have a chance to start enjoying traveling again. And by that I mean having the time to take a long distance train, and even better, to ditch those bad imitations of birds with nicer long distance, transatlantic boats taking 10 days to take us over the oceans.

Imagine boarding in Genova or Hamburg, Le Havre or Cadiz, and taking your time to go through the Atlantic again. Let's be clear, this is not 1920; with email, Skype or iChat you won't miss much in terms of meetings or anything, but you'll get to New York or Boston without jet lag, relaxed, sipping a margarita on the main bridge while waving to the people on the shore.

I would enjoy it for sure. And if the Eyjafjallajökull[4] (somebody please tell me how to pronounce this) keeps on spitting ashes, at least until the Jet Stream cleans up the stratosphere a bit over the northern Atlantic, we won't have any other option, anyway.

In the meantime, let's relax and enjoy the first spring with a really, really blue sky, without airplanes or long white smoke trails, in a long, long time.

[4] http://en.wikipedia.org/wiki/2010_eruptions_of_Eyjafjallajökull

How to buy iPad apps outside of the US

Adrian Kosmaczewski

2010-04-21

So you flew and bought an iPad, you live outside of the US, and you would like to buy iBooks[1], Pages[2], Numbers[3] and Keynote[4] (as well as many other amazing apps) from your own country, huh? After all, as Scoble said[5], "the iPad without apps is pretty lame, actually".

Well, welcome to the club[6]. Here's a couple of tips to get you started and buy applications that are only available in the US:

1. First, I assume you have a valid iTunes (App Store) login for your local store, whichever the country. You can browse and buy iPhone applications in iTunes in the Mac where you usually synchronize your iPhone or, now, your iPad.
2. The App Store on the iPad is meant to choose and buy iPad applications; however, currently only the US App Store allows to "browse" those applications. To browse and buy applications in

[1] http://www.apple.com/ipad/features/ibooks.html
[2] http://www.apple.com/ipad/features/pages.html
[3] http://www.apple.com/ipad/features/numbers.html
[4] http://www.apple.com/ipad/features/keynote.html
[5] http://scobleizer.com/2010/04/14/why-was-apples-prediction-on-ipads-so-wrong/
[6] /blog/first-ipads-in-switzerland/

the US App Store you need an account... without credit card information! Apple has published a support article[7] that contains all the required information to do this.

3. Using the account created in the previous step, you can buy all free iPad apps available in the US App Store. This includes the amazing iBooks[8] app, which I strongly recommend you to buy. However, without credit card information, you cannot buy paying apps. **However**, you can buy them from iTunes in your own country. The trick is to search for them in the search engine :) that's it! Even if you cannot **browse** for iPad apps, that does not mean that they are not there; many of them might just as well be available in your country, right now! To find them, just search them using their name in the search box, and buy them with your local country App Store (iTunes) account. Yup, you can do this. Then, sync your iPad with your Mac, and you'll have those apps installed in a snap.
4. However (there's always a "but") you cannot buy Pages[9], Numbers[10] or Keynote[11] for iPad this way, because Apple has not yet released those apps in other countries than the US. For that, you can buy an iTunes gift card[12]! I have bought two 50$ cards with this company, and they just rock! I know other 4 people who used their services with 100% satisfaction, so go on, and as soon as you receive the e-mail with the gift card codes, enter them **in the App Store in your iPad** (not in your Mac, OK?) and you'll be able to buy the three productivity apps, plus a couple[13] of games[14], why not?

You can now sit on your couch[15] and enjoy your iPad full of apps!

[7] http://support.apple.com/kb/HT2534
[8] http://www.apple.com/ipad/features/ibooks.html
[9] http://www.apple.com/ipad/features/pages.html
[10] http://www.apple.com/ipad/features/numbers.html
[11] http://www.apple.com/ipad/features/keynote.html
[12] http://itunes-giftcards.com/Home.html
[13] http://ax.itunes.apple.com/us/app/real-racing-hd/id363998989?mt=8&ign-impt=clickRef%3DSoftware%2520Page-US-Real%2520Racing%2520HD-363998989-Lockup
[14] http://itunes.apple.com/us/app/scrabble/id284815117?mt=8
[15] /blog/the-dawn-of-couch-computing/

initWithContentsOfURL: Methods Considered Harmful

Adrian Kosmaczewski

2010-05-28

As I promised on Twitter[1], here's a small discussion about the problems brought by the "initWithContentsOfURL:" family of methods.

[1] http://twitter.com/akosma/status/14715706200

A quick search in the Xcode documentation browser brings in an interesting list of classes including this initializer (with or without additional parameters):

- NSArray
- NSManagedObjectModel
- NSData
- NSDictionary
- NSXMLParser

- NSMappingModel
- NSString
- AVAudioPlayer

Don't get me wrong, the title of this post is a bit misleading; it's not that this method is always problematic, particularly when the URL passed as parameter has been built using the fileURLWithPath: family of initializers. In this case, the URL will point to a local resource, and the worst thing that can happen in that case is that the filename is wrong, or that the file does not exist for some reason (NSFileManager to the rescue!).

However, when using external URLs referencing resources somewhere over the network, **using this method is definitely a recipe for disaster.**

The main problem with these methods, of course, is the fact that they are **synchronous**; this means that the thread executing them (usually the UI thread) will block completely until they return, and in most applications this means that you are de-facto blocking the whole application for an unknown amount of time. This means that no buttons or UI widgets will react to input, no navigation will be possible, no touch events will be delivered or executed, nothing will happen at all until the network operation completes.

Even worse; when using initWithContentsOfURL:, there is no timeout, there is no meaningful feedback for network failures, and no way for the user to cancel the current network operation. This last factor justifies by itself not using initWithContentsOfURL: at all; you must never ship code that leads to a bad user experience. Your users will resent this and will complain!

As Joel Spolsky explained in his classic article Three Wrong Ideas From Computer Science[2],

> One example of network transparency is the famous RPC (remote procedure call), a system designed so that you can call procedures (subroutines) running on another computer on the network exactly as if they were running on the local computer. An awful lot of energy went into this. Another example, built on top of RPC, is Microsoft's Distributed COM (DCOM), in which you can access objects running on another computer as if they were on the current computer.
>
> Sounds logical, right?
>
> Wrong.
>
> There are three very major differences between accessing resources on another machine and accessing resources on the local machine:
>
> 1. Availability,

[2] http://www.joelonsoftware.com/articles/fog0000000041.html

2. Latency, and
3. Reliability.

(BTW, read this article, it's really a good one, like many others in his blog)

In summary, never assume that the network is available, never assume that anything behind the network is available, and never assume that a network has any particular speed. This is particularly true in the case of mobile platforms, where network conditions can vary tremendously from one minute to the other!

The problem with the initWithContentsOfURL: methods is that some online tutorials use them as a quick-and-dirty way to load resources from the network (for example this one on iCodeBlog[3]), and I have myself seen production code using this method.

This is very, very bad. Using initWithContentsOfURL: with remote network resources in a tutorial like this is simply a horrendous, almost irresponsible approach towards developers new to the iPhone platform.

You must not use synchronous connections in any way in your code, unless you are 100% sure that you are running it in a non-UI thread – and even so, as Jeff LaMarche explained recently[4], you should avoid such multithreaded approaches. The NSRunLoop architecture of Cocoa and Cocoa touch allows you to bypass threading altogether when dealing with networking operations. Remember this! And read Jeff's article for more details.

This exact same problem can also be created with other APIs and libraries, for example when using NSURLConnection's sendSynchronousRequest:returningResponse:error: method (as explained today in The Apple Blog[5].)

Do not block the main UI thread. Write this on your whiteboard, on a post-it on your computer monitor, in your agenda with a reminder every day. Do not use this approach. As Cédric said[6], these methods should only be used for prototyping!

One popular example of a network library that support asynchronous connections is ASIHTTPRequest[7], which I tend to use in nearly all my projects (instead of the standard NSURLConnection classes bundled in Cocoa and Cocoa Touch.) I prefer it because it has a nicer, smaller interface, it's fast (I've benchmarked it in my iPhone Web Services

[3] http://icodeblog.com/2009/06/19/using-nsxmlparser-to-pull-uiimages-from-the-web/

[4] http://iphonedevelopment.blogspot.com/2010/05/downloading-images-for-table-without.html

[5] http://theappleblog.com/2010/05/28/iphone-dev-sessions-responsive-web-enabled-iphone-apps/?utm_source=feedburner&utm_medium=feed&utm_campaign=Feed%3A+TheAppleBlog+%28TheAppleBlog%29

[6] http://twitter.com/0xced/status/14717929660

[7] http://allseeing-i.com/ASIHTTPRequest/

Client[8] demo project on Github), and it provides handy queueing capabilities. FTW!

By the way, Ben Copsey (the creator of ASIHTTPRequest) has published today an article in his blog[9], in which he exposes the approaches used to queue network connections in his library. Designing a library for concurrent, queued network connections is not trivial, and his current approach (subclassing NSOperationQueue) might soon be deprecated for a new, NSRunLoop-y, more Cocoa-friendly, way.

In any case, I hope I have made my point: avoid synchronous operations when loading content from the network, remember what Joel said, and use a library providing asynchronous connections. Happy coding!

PS: by the way, the title of this blog post is a reference to many other classic articles, so many that there's even a Wikipedia entry about them[10] and a great rant about them[11] by Eric Meyer!

[8] http://github.com/akosma/iPhoneWebServicesClient

[9] http://allseeing-i.com/A-possible-replacement-for-ASINetworkQueue

[10] http://en.wikipedia.org/wiki/Considered_harmful

[11] http://meyerweb.com/eric/comment/chech.html

Objective-C Categories as Stylesheets

Adrian Kosmaczewski

2010-06-03

It is very important that iPhone and iPad applications use visual styles in a coherent way. This helps users learn how to use your application faster, it helps them scan your UI for important information as quickly as possible, and it also can convey a strong marketing message; companies who want iPhone or iPad applications often have complex visual identities, including predefined fonts and colors, and they will want their applications to match those choices.

However, getting all the UI widgets to look similarly can be complex, particularly in large applications; what if your client or their designers change their minds about the font size or some background color right before shipping your project? Of course you can "search and replace" all occurrences of some color using Xcode, but you have the risk of leaving some unchanged widget somewhere. And believe me, this happens really often.

In this article I will discuss a simple approach, using Objective-C categories, to keep your styling information separated from the rest of the application, using a system that will be familiar to developers used to creating websites using CSS (Cascading Style Sheets).

First Approach: Simple Categories

The first, easiest approach is to create simple UIFont and UIColor categories in your application, and provide semantic descriptions of the data you want to style; for example "customerNameFont" and "companyPhoneColor" are good names. You can add as many class methods as you want to the UIFont and UIColor classes for that:

```
@interface UIFont (YourAppName)

+ (UIFont *)customerNameFont;
+ (UIFont *)customerPhoneFont;

@end

@implementation UIFont (YourAppName)

+ (UIFont *)customerNameFont
```

```
{
    return [UIFont fontWithName:@"Helvetica" size:22.0];
}

+ (UIFont *)customerPhoneFont
{
    return [UIFont boldSystemFontOfSize:15.0];
}

@end
```

And we provide the same treatment to the UIColor class:

```
@interface UIColor (YourAppName)

+ (UIColor *)customerNameColor;
+ (UIColor *)customerPhoneColor;

@end

@implementation UIColor (YourAppName)

+ (UIColor *)customerNameColor
{
    return [UIColor blueColor];
}

+ (UIColor *)customerPhoneColor
{
    return [UIColor redColor];
}

@end
```

Then, you can use those method names in the rest of your application:

```
- (UITableViewCell *)tableView:(UITableView *)tableView cellForRowAtIndexPath:(
{
    static NSString *CellIdentifier = @"Cell";

    UITableViewCell *cell = [tableView dequeueReusableCellWithIdentifier:CellId
    if (cell == nil)
    {
        cell = [[[UITableViewCell alloc] initWithStyle:UITableViewCellStyleSubt
                                        reuseIdentifier:CellIdentifier] autorele

        // And here goes the styling!
        cell.textLabel.font = [UIFont customerNameFont];
        cell.textLabel.textColor = [UIColor customerNameColor];

        cell.detailTextLabel.font = [UIFont customerPhoneFont];
```

```
        cell.detailTextLabel.textColor = [UIColor customerPhoneColor];
    }

    Customer *cust = [self.data objectAtIndex:indexPath.row];
    cell.textLabel.text = cust.name
    cell.detailTextLabel.text = cust.phone

    return cell;
}
```

The advantage of this simple approach is that all styles are managed from a central location, and you can change the fonts and colors of your application in a single step.

There are, however, two major disadvantages:

1. Colors and fonts defined on their respective categories will not be available in Interface Builder, so you might have to override viewDidLoad or other methods to force the styling at a certain moment, or to set the color and font values manually, which might lead to duplicate information.
2. Some UI elements have support for even more styling information, like shadows or CGAffineTransform values, which cannot be handled in a simple way with this method.

We need some kind of CSS for iPhone applications, and the next section provides a rudimentary approach to that problem.

Second Approach, “Cascading Style” Classes

Let’s push the idea of semantic styling a bit; in the previous example, we created separated categories for UIFont and UIColor and used them in atomic form, changing individual properties of your widgets, one by one.

What if you wanted to set several properties at once, just like with CSS classes on web pages? What if you had a system which could be extended to support more properties, for other, as of yet unknown, future UI widgets? For that, we would need a special container for style information, like a CSS stylesheet, and we should be able to assign this container to any kind of visual widget; in turn, those widgets would automatically adapt their layout and appearance. There are many different ways to do this; and this is just one of them.

First I declare a class called AKCascadingStyle, which holds basic styling information, and which is able to apply that information to any kind of object passed in parameter:

```
@interface AKCascadingStyle : NSObject
{
@private
    UIFont *_font;
    UIColor *_textColor;
```

```
    UIColor *_backgroundColor;
    UIColor *_tintColor;
}

@property (nonatomic, retain) UIFont *font;
@property (nonatomic, retain) UIColor *textColor;
@property (nonatomic, retain) UIColor *backgroundColor;
@property (nonatomic, retain) UIColor *tintColor;

+ (id)style;
+ (id)styleFromObject:(id)object;

- (void)applyToObject:(id)object;
- (void)setValuesFromObject:(id)object;

@end
```

This class can be inherited and extended, and its methods can be overridden, as we'll see shortly, to add support for more properties. Then I add a category on NSObject (careful with this!) to support adding and retrieving style information in the form of AKCascadingStyle instances:

```
@class AKCascadingStyle;

@interface NSObject (AKCascadingStyle)

@property (nonatomic, retain) AKCascadingStyle *cascadingStyle;

@end
```

Why a category on NSObject? Because not everything you see on your iPhone screen is a subclass of UIView! UIBarButtonItems, for example, inherit from UIBarItem, which itself inherits from NSObject, and not from UIView. By the way, by extending NSObject, this code could also be used in Mac OS X applications, for example, without modification.

Given that categories cannot add ivars to existing classes, the getter of this property will call the [AKCascadingStyle styleFromObject:] method above:

```
@implementation NSObject (AKCascadingStyle)

@dynamic cascadingStyle;

- (void)setCascadingStyle:(AKCascadingStyle *)style
{
    [style applyToObject:self];
}

- (id)cascadingStyle
```

```
{
    return [AKCascadingStyle styleFromObject:self];
}

@end
```

Both [AKCascadingStyle applyToObject:] and [AKCascadingStyle styleFromObject:] are polymorphic (as all Objective-C methods are) so subclasses can extend their functionality as required (but don't forget to send the same message to "super" before!):

```
@interface ShadowStyle : AKCascadingStyle
{
@private
    CGSize _shadowOffset;
    UIColor *_shadowColor;
}

@property (nonatomic) CGSize shadowOffset;
@property (nonatomic, retain) UIColor *shadowColor;

@end
```

Then, to use these style classes in your own controllers, just do the following:

```
- (IBAction)addStyle:(id)sender
{
    self.contentsTextView.cascadingStyle = [ShadowStyle style];
}
```

Setting this property will automatically set all the required parameters in your widget; text color, background colors, transformations, you name it. You can inherit styles in order to reuse them, and you can style pretty much any kind of UIKit widget with it.

You can download the code of this project from Github[1], as usual! Enjoy!

[1] http://github.com/akosma/AKCascadingStyle

WWDC 2010

Adrian Kosmaczewski

2010-06-04

Folks, it's that time of the year once again: WWDC 2010 is around the corner![1]

This will be my third time in San Francisco to attend the ultimate conference for Mac, iPhone (and now iPad!) developers, and to learn the latest and greatest from Apple. I won't be giving any advice about it, since Jeff LaMarche[2], Mike Lee[3] and Marco Arment[4] and have done a great job at it. Be sure to check out their articles before going to SF.

[5]

Oh, well, maybe yes, I'll be giving one particular advice: don't forget a pullover or sweater, particularly if you are going to queue for the keynote early on Monday weather in SF is particularly cold on nights and mornings, even during June! I can't stress how cool it is to be there, how many interesting people you get to meet, and how many things you get to learn. Every year the investment has paid off!

In any case, be sure to check out the session and labs calendar[6] (login

[1] http://developer.apple.com/wwdc/

[2] http://iphonedevelopment.blogspot.com/2010/04/wwdc-first-time-guide-2010-edition.html

[3] http://www.atomicwang.org/motherfucker/Index/Entries/2010/5/23_The_mean_streets_of_San_Francisco.html

[4] http://www.marco.org/661870733

[5] http://developer.apple.com/wwdc/

[6] https://developer.apple.com/wwdc/sessions/

required) and don't miss the (un)official list of WWDC parties[7] and evening events. I will be attending some of them too.

Also, on Sunday I'll be going to Cupertino in the field trip organized by Jeff[8] and Scott Knaster[9], probably taking the 11am bus from Moscone South. If you are around, be sure to stop me to say hi! I'd love to meet you in person.

PS: Ramin Firoozye has just published a visitor's guide to San Francisco[10] which might also be of help.

PS2: More tips from Daniel Pasco[11] from Black Pixel[12] and from Brent Simmons[13]!

[7] http://wwdcparties.com/

[8] http://iphonedevelopment.blogspot.com/2010/05/pre-wwdc-pilgrimage-sign-up.html

[9] http://www.knaster.com/

[10] http://ramin.firoozye.com/2010/06/04/a-visitors-guide-to-wwdc-and-san-francisco/

[11] http://softarts.tumblr.com/day/2010/06/05

[12] http://blackpixel.com/

[13] http://inessential.com/2010/06/05/practical_wwdc_tips

Applications iPhone: il est temps de s'y mettre!

Adrian Kosmaczewski

2010-06-05

Le prochain mercredi 16 juin j'aurai le privilège de partager la scène avec Marco Scheurer de Sen:te[1], Surane Ragavan de abonobo[2], et Bertrand Baeriswyl lors du séminaire First "Applications iPhone: il est temps de s'y mettre!"[3] organisé par Rezonance[4], créateur de liens en Suisse Romande:

rezonance
Créateur de liens en Suisse romande

> Plus de 140'000 applications, 3 milliards de téléchargements, 50 millions de propriétaires d'iPhones, dont 500'000 en Suisse… L'importance prise par l'iPhone et ses multiples applications est considérable. De nombreuses entreprises ont créé leur application iPhone pour se rendre accessibles à leurs consommateurs/-trices partout et à tout moment.
>
> Comment développer une application iPhone avec succès et à succès? Quels sont les challenges techniques, ergonomiques, financiers ? Quels sont les enjeux de la création d'une application pour une entreprise ? Avec quels résultats ? Pourquoi développer une application iPhone plutôt qu'une référence sous Android ou Windows Mobile ? Quelles différences entre une page web mobile ou une application dédiée ?
>
> Ce First est organisé dans le cadre du Master of Advanced Studies in Information and Communication Technologies (MAS-ICT) des cinq Hautes Ecoles d'ingénierie de la HES-SO, du Master of Advanced Studies in Rapid Application Development (MAS-RAD) de la HES-SO et du nouveau Certificate of Advanced Studies en production d'applications

[1] http://www.sente.ch/

[2] http://www.abonobo.com/

[3] http://www.rezonance.ch/rezo/classes/ft-first-tuesday/vaud/20100616/one-community?page_num=0

[4] http://www.rezonance.ch/

interactives (CAS-PAI) du Département Formation Continue de la HEIG-VD.

Je me réjouis de vous y trouver et de parler iPhone et iPad avec vous!
Merci beaucoup à Rezonance pour organiser un tel évènement.

Rezonance First: Panorama et enjeux du marché des applications iPhone

Adrian Kosmaczewski

2010-06-17

Voici les slides de la présentation que j'ai donné hier dans le cadre du Rezonance First "Applications iPhone: il est temps de s'y mettre!"[1]

Merci à l'équipe de Rezonance pour l'organisation impeccable, à l'HEIG VD de Yverdon pour l'accueil magnifique et a tous ceux qui sont venus!

[1] http://www.rezonance.ch/rezo/classes/ft-first-tuesday/vaud/20100616/one-community?page_num=0

“Dev Day for iPhone” sur Le Temps

Adrian Kosmaczewski

2010-06-28

> «Face à cette demande, il manque des développeurs pour iPhone en Suisse. Nous devons être une centaine en Suisse romande et n’arrivons pas à répondre à toutes les offres» confirme Stephan Burlot, de la société Coriolis, à Chevilly (VD). «Bon, de nombreux développeurs viennent du monde de l’open source et ne sont pas forcément à l’aise avec le monde plus fermé d’Apple», sourit Adrian Kosmaczewski, directeur d’Akosma Software, à Pully.

“Pour l’iPhone, il manque des développeurs en Suisse”[1], interview par Anouch Seydtaghia publiée le 24 juin dernier sur Le Temps[2].

[1] http://letemps.ch/Page/Uuid/08633638-7f07-11df-b539-5d63add52629/Pour_liPhone_il_manque_des_développeurs_en_Suisse

[2] http://letemps.ch/

Tweeting Without Twitter

Adrian Kosmaczewski

2010-07-01

During my flight to WWDC this year I could not really sleep, and the 12 hour flight was the source of memorable tweets that will never make it to Twitter. Because of timing and context, and also because of the inexcusable lack of wifi network in some major airlines.

Anyway.

I used Pages during the flight to keep track of all those insomniac, bilingual tweets, while the plane was a going through the Atlantic and Canada towards San Francisco. Some are about the flight itself, others about the Argentine film “El Secreto de sus Ojos”, and finally some about the Football World Cup. Project yourself in the situation, and enjoy the rants. I certainly did :)

During and about the flight

- The iPad is the perfect onboard entertainment system. Pages as an offline Twitter client, Kindle and iBooks as ebook readers, some movies, and lots of music. 57% battery after 12 hours.
- As the stewards would say: “brace, brace”; lots of tweets coming.
- Outlook is a city in Saskatchewan, not far from Saskatoon and Moose Jaw. After this revelation to remember the next time you play Trivial Pursuit, you can resume your normal activities.
- Saskatoon officially gets the 2010 akosma award for the best city name ever.
- As you could imagine, the only interesting thing in this inflight entertainment thingy is the 3D map indicating our current position. #fascinating #boring
- I forgot to mention that my seat’s sound system is broken. I could watch any movie I want but my lip reading skills are not that good.
- The names of the cities in northern Canada are just amazing. Can’t remember any of them tho. Are they Innuit names?
- Our vision of the world is as distorted as a map of Novaja Zemlya in Mercator projection after the explosion of the Tsar bomba.
- Why do they still print the “smoke” section in boarding passes? Is there any airline out there still offering smoking seats?

- Given that most airplane tickets are electronic, why aren't boarding passes? #iphoneappidea
- Let's calculate the CO2 emissions caused by airlines still printing the word "smoke" in boarding passes. No, let's better not.
- The keyboard of the iPad has this character: â,© (tap and hold the dollar sign). What currency is it? Korea's won? If Korea won, then the pun is intended.
- On the plane with @mediaatelier and @dcondrau, probably even more Swiss devs, but without wifi and Twitter, difficult to know.
- There were fewer devs using Xcode on this Swiss flight than in last year's Lufthansa flight. I will leave the elaboration of any conclusion thereof to my dear followers.
- Why do flight attendants akways decide to serve beverages at precisely the same moment when planes go through turbulences? #complot #midwest #twister
- Now I understand why Swiss is Lufthansa's cash cow: pricier tickets and crappier service. Only selling point: the nonstop ZRH -> SFO flight. #swissairwherearethou
- I remember when they added phones to Swissair airplanes, back in 1996; calls used to cost 10 dollars per minute. Now, in 2010, in Swiss… they cost the same. #WTF
- In the airport of Santa Cruz de la Sierra (Bolivia) there was free, fast, open wifi in 2006. Most airports in the northern hemisphere, in 2010, charge a lot for a crappy connection. #WTF
- And still, no wifi in planes. #WTF
- No, I haven't been involved in Nazi activities during from 1933 to 1945. Thanks for asking. #visawaiver
- Swiss people not traveling often outside of Switzerland are easy to spot. I let you imagine the rest of this tweet.
- The Swiss version of Homer Simpson is sitting beside me, and is kinda fascinated with my iPad. Yeah, I'm talking about you, moron.
- Oh no my dear Homer, the armrest between us is mine for the rest of the flight. See? I don't push my elbow on your side. Wasn't that hard, now was it?
- The "Skytrain" in ZRH airport, is a subway, actually. You can hear yodel and cows and other Swiss sounds inside while you go from terminal to terminal. #typisch
- Whoever said that the iPad is a consumption-only device should have stop consuming some substances before writing nonsense.

Sobre la pelicula "El Secreto de sus Ojos"

- El otro día fui a ver "El Secreto de sus Ojos" con Clau, y me quedaron, obviamente, muchas cosas picando, pensaturrias.
- Me imagino la Argentina de mis padres como un lugar con una dosis mayor de inocencia de la actual. Debe haber haber sido un lindo lugar.
- Vivir en la Argentina de los 80 no estuvo mal. Aparte de las

hiperinflaciones y crisis crónicas, yo tengo lindos recuerdos de aquella época.

Sobre el mundial

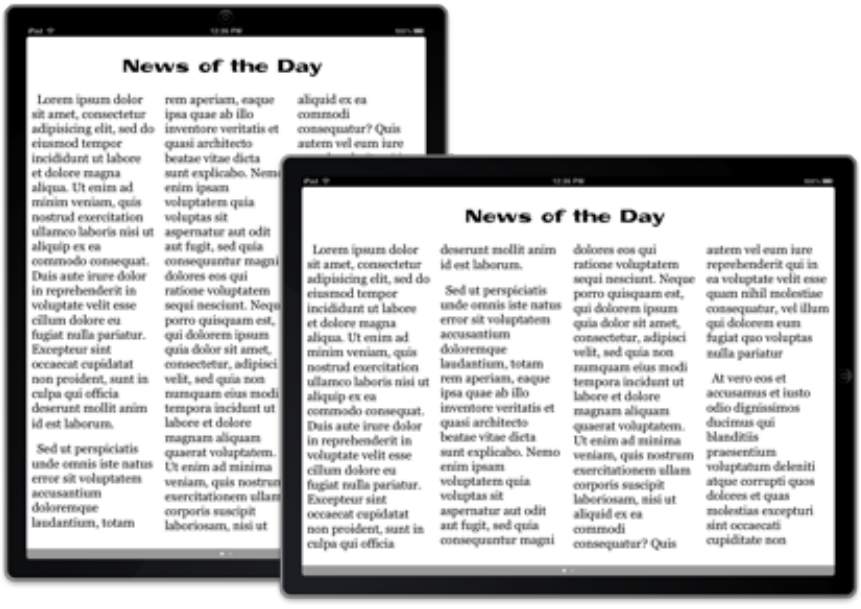
- Si Drogba se fracturo y quedo afuera del mundial, entonces se queda con las Ghana. Y el resto del tweet es superfluo. #mundial
- Si te fracbturas, necesitas una drogba para conbtener el dbolor. #mundial
- Primero Beckham, después Rooney, ahora Drogba... que otros jugadores se quedaron afuera? #mundial
- Atacante norcoreano inscripto como arquero... te imaginas? El técnico lo tendría que probar al arco, por ahí después se lo recordaría como el Higuita coreano. #escorpión #taekwondo #mundial
- El conocido árbitro paraguayo Carlos Amarilla no participará en el mundial. Tampoco asistirán el reconocido juez de línea francés Marcel Orsay ni el legendario hincha de Camerún. #burumbumbúm #mundial
- Burumbumberia, burumbumberia, yo soy el hincha, de Nigeria. #mundial
- Burumbumbana, burumbumbana, yo soy el hincha, de Ghana. #mundial
- Burumbumbafrica, burumbumbumbafrica, yo soy el hincha, de Sudafrica. #mundial
- Burumbumbar, burumbumbar, yo soy el hincha, de Madagascar (las hinchadas de Zanzibar y Escobar cantan similares canciones) #mundial
- Burumbumbique, burumbumbique, yo soy el hincha, de Mozambique. #mundial
- Burumbumipto, burumbumipto, yo soy el hincha, de ..... (enviar respuesta correcta en un reply) #mundial
- Burumbumbina, burumbumbina, yo soy el hincha, de Argentina (el que diga “efedrina” es boleta). #mundial
- Burumbumbal, burumbumbal, yo soy el hincha, de Codesal (cantito alemán) #mundial
- Burumbumbola, burumbumbola, yo soy el hincha, de Angola. #mundial
- Burumbumbasta, burumbasta, yo creo... que ya basta.
- El otro día dieron en la TV Suiza (en italiano) la película de Kusturica sobre Maradona. Muy buena.ï»¿

Core Text Objective-C Wrapper

Adrian Kosmaczewski

2010-07-08

One of the most promising and mysterious new frameworks introduced in iOS 3.2 is Core Text. Apple defines Core Text as a “text drawing engine”, which allows Mac (and now iPad) apps to render rich text on any graphics context. Strings drawn with Core Text feature lots of custom settings such as detailed font information, columns, variable line and paragraph heights and several different attributes, which designers and font aficionados surely understand much better than I do. Many new apps have been using this framework since the release of the iPad, particularly newspapers and ebook reader applications, rendering gorgeous text in custom fonts, many of them not available natively in iOS. This framework is also used in lifestyle and corporate applications, too, where using a custom font is sometimes required to match the specifications of brands and trademarks.



It is very important to understand that Core Text is really just a text drawing engine to be used on top of Quartz (Core Graphics), to render rich text on any graphics context. Core Text cannot be used to create a rich text editor, for example, so don’t expect to extend UITextView

with it. But you can use it to draw any kind of rich text on screen, which can make you avoid using UIWebView instances for that.

One of the most interesting capabilities of Core Text is being able to render text in several columns. However, Core Test is a C-based framework, and I think that understanding and using the concepts and structures required to render text in columns can be particularly tricky. To make the my life and that of my fellow developers easier, I've created a small set of Objective-C classes that encapsulate the creation of framesetters, attributed strings and other constructions, and takes care of the creation of several columns, as well as the division of the text in several pages if required.

The API interface is very simple (in purpose) and I'm pretty sure you'll be able to integrate it very easily in your own projects, particularly if you look at the sample project where the classes are used. I've also added a category for UIFont, that returns the associated CTFontRef pointer, in a similar fashion to UIImage, which is able to return a pointer to the underlying CGImageRef pointer. It also allows to create a CTFontRef from any font embedded in the application bundle. I am puzzled that the framework designers haven't included this by default in UIKit.

A future extension I'd like to add would be a couple of categories to parse simple RTF or HTML strings (to start with, probably just with bold and italic text) and create the appropriate attributed string from it; there's a couple of AppKit extensions to NSAttributedString that do exactly that, but for the moment they are only available in the Mac version of Cocoa, and I haven't found anything similar for iOS yet (feel free to leave a comment below if you know about some implementation for generating NSAttributedStrings from HTML or RTF text!)

The code, as usual, is available through Github[1] (BSD license), so feel free to use it and even contribute bug fixes or enhancements if you want. The font embedded in the project is Polsku Regula, available through the Open Font Library.org[2]. The project requires Xcode 3.2.3 and the iOS SDK 4.0, and it creates a simple application that works on the iPad (iOS 3.2). As usual, use it at your own risk. Enjoy!

[1] http://github.com/akosma/CoreTextWrapper
[2] http://openfontlibrary.org/files/ospublish/140

digital2.0 iPad Application

Adrian Kosmaczewski

2010-07-09

This is a very special blog post to introduce our latest project, a unique collaboration between moser[1], a brand and design specialist, VPS prod[2], an award-winning video production company, and akosma software:

[3]

digital2.0[4], available on the App Store, is an exploration into the capabilities of the iPad. Everybody talks about this new device, but what can you do with it? We hear this question all the time, from our clients and even from our colleagues and the teams we work with. So we decided to join forces and create a demo application answering most of the questions raised by this new type of media.

digital2.0[5] provides answers to the following questions:

- How does the iPad deal with changes in the orientations of the device?
- Which fonts are available in the device? Can I add my own corporate fonts?
- What kind of interactivity is available in the iPad?

[1] http://www.moserdesign.ch/
[2] http://vpsprod.com/
[3] http://itunes.com/apps/digital20
[4] http://itunes.com/apps/digital20
[5] http://itunes.com/apps/digital20

- How do you integrate video content into an iPad application?
- Can you retrieve and display real-time data on an iPad?
- What about 3D? How do you integrate it into an application?
- Does the iPad know where the user is located?
- How can you connect the iPad to social networking sites?

Finally, we've also included a "Making of" video, shot during the 3 months of work required to create this project since April. Around 10 people were involved in this project, from graphic designers to video directors, and we are all really proud of the result. We hope you'll like it too!

[6]

If you want to get more information about digital2.0, check out the official Facebook page[7] of the project and also our Twitter account[8].

[6] http://itunes.com/apps/digital20
[7] http://www.facebook.com/pages/digital20/122084417836409
[8] https://twitter.com/digital2dot0

Making of digital2.0

Adrian Kosmaczewski

2010-07-10

This is the video[1] shot by the amazing team of VPS prod[2] while we were working on the application. There was always a camera rolling during our meetings and even during our work sessions! The final result is simply outstanding and we are very happy to release it in public.

The videos included in the digital2.0[3] application are available in HD format from YouTube[4] and from Vimeo[5]. Check them out!

[1] https://www.youtube.com/watch?v=B91_JXEoSn4
[2] http://vpsprod.com/
[3] http://itunes.com/apps/digital20
[4] http://www.youtube.com/user/digital2dot0
[5] http://vimeo.com/digital2dot0

nib2objc updated

Adrian Kosmaczewski

2010-07-17

I've just committed version 1.2 of nib2objc[1] to Github with the following enhancements:

1. Compatibility with Xcode 3.2.3 and the latest iPhone SDKs;
2. In debug mode, the tool generates stub comments for properties not currently recognized;
3. Provides much nicer variable names; instead of "view24" you get "button53" or "tableview34"; those names are also used in the hierarchy output at the end of the generated code;
4. Added support for MKMapView instances;
5. Added support for some new UIView properties: autoresizesSubviews and contentStretch;
6. Added support for UITextView's clearButtonMode property;
7. Added support for UIImageView's highlighted property;
8. Added support for the following UISearchBar properties: showsScopeBar, showsSearchResultsButton, scopeButtonTitles;
9. Added support for the following UITableViewCell properties: editingAccessoryType, shouldIndentWhileEditing;
10. Added support for the following UITableView properties: rowHeight, sectionFooterHeight, sectionHeaderHeight;
11. Removed code generated for deprecated API calls (since SDK 3.0) in UIButton and UITableViewCell;

In particular, item #3 above is my preferred: the output of a simple NIB files looks like this now:

```
UIActivityIndicatorView *activityindicatorview8 = [[UIActivityIndicatorView all
activityindicatorview8.frame = CGRectMake(309.0, 669.0, 20.0, 20.0);
activityindicatorview8.alpha = 1.000;
activityindicatorview8.autoresizesSubviews = YES;
activityindicatorview8.autoresizingMask = UIViewAutoresizingFlexibleRightMargin
activityindicatorview8.clearsContextBeforeDrawing = YES;
activityindicatorview8.clipsToBounds = NO;
activityindicatorview8.contentMode = UIViewContentModeScaleToFill;
activityindicatorview8.contentStretch = CGRectFromString(@"{{0, 0}, {1, 1}}");
activityindicatorview8.hidden = NO;
activityindicatorview8.hidesWhenStopped = NO;
activityindicatorview8.multipleTouchEnabled = NO;
```

[1] http://github.com/akosma/nib2objc

```
activityindicatorview8.opaque = NO;
activityindicatorview8.tag = 0;
activityindicatorview8.userInteractionEnabled = YES;
[activityindicatorview8 stopAnimating];

UISegmentedControl *segmentedcontrol6 = [[UISegmentedControl alloc] initWithIte
segmentedcontrol6.frame = CGRectMake(174.0, 466.0, 207.0, 44.0);
segmentedcontrol6.alpha = 1.000;
segmentedcontrol6.autoresizesSubviews = YES;
segmentedcontrol6.autoresizingMask = UIViewAutoresizingFlexibleRightMargin | UI
segmentedcontrol6.clearsContextBeforeDrawing = YES;
segmentedcontrol6.clipsToBounds = NO;
segmentedcontrol6.contentHorizontalAlignment = UIControlContentHorizontalAlignm
segmentedcontrol6.contentMode = UIViewContentModeScaleToFill;
segmentedcontrol6.contentStretch = CGRectFromString(@"{{0, 0}, {1, 1}}");
segmentedcontrol6.contentVerticalAlignment = UIControlContentVerticalAlignmentT
segmentedcontrol6.enabled = YES;
segmentedcontrol6.hidden = NO;
segmentedcontrol6.highlighted = NO;
segmentedcontrol6.momentary = NO;
segmentedcontrol6.multipleTouchEnabled = NO;
segmentedcontrol6.opaque = NO;
segmentedcontrol6.segmentedControlStyle = UISegmentedControlStylePlain;
segmentedcontrol6.selected = NO;
segmentedcontrol6.selectedSegmentIndex = 0;
segmentedcontrol6.tag = 0;
segmentedcontrol6.userInteractionEnabled = YES;

UIImageView *imageview4 = [[UIImageView alloc] initWithFrame:CGRectMake(79.0, 4
imageview4.frame = CGRectMake(79.0, 47.0, 311.0, 260.0);
imageview4.alpha = 1.000;
imageview4.autoresizesSubviews = YES;
imageview4.autoresizingMask = UIViewAutoresizingFlexibleWidth | UIViewAutoresiz
imageview4.clearsContextBeforeDrawing = YES;
imageview4.clipsToBounds = NO;
imageview4.contentMode = UIViewContentModeScaleToFill;
imageview4.contentStretch = CGRectFromString(@"{{0, 0}, {1, 1}}");
imageview4.hidden = NO;
imageview4.highlighted = NO;
imageview4.multipleTouchEnabled = NO;
imageview4.opaque = YES;
imageview4.tag = 0;
imageview4.userInteractionEnabled = NO;

UIView *view2 = [[UIView alloc] initWithFrame:CGRectMake(0.0, 0.0, 768.0, 1004.
view2.frame = CGRectMake(0.0, 0.0, 768.0, 1004.0);
view2.alpha = 1.000;
view2.autoresizesSubviews = YES;
view2.autoresizingMask = UIViewAutoresizingFlexibleWidth | UIViewAutoresizingFl
view2.backgroundColor = [UIColor colorWithWhite:1.000 alpha:1.000];
```

```
view2.clearsContextBeforeDrawing = YES;
view2.clipsToBounds = NO;
view2.contentMode = UIViewContentModeScaleToFill;
view2.contentStretch = CGRectFromString(@"{{0, 0}, {1, 1}}");
view2.hidden = NO;
view2.multipleTouchEnabled = NO;
view2.opaque = YES;
view2.tag = 0;
view2.userInteractionEnabled = YES;

UISlider *slider7 = [[UISlider alloc] initWithFrame:CGRectMake(520.0, 588.0, 11
slider7.frame = CGRectMake(520.0, 588.0, 118.0, 23.0);
slider7.alpha = 1.000;
slider7.autoresizesSubviews = YES;
slider7.autoresizingMask = UIViewAutoresizingFlexibleRightMargin | UIViewAutore
slider7.clearsContextBeforeDrawing = YES;
slider7.clipsToBounds = NO;
slider7.contentHorizontalAlignment = UIControlContentHorizontalAlignmentCenter;
slider7.contentMode = UIViewContentModeScaleToFill;
slider7.contentStretch = CGRectFromString(@"{{0, 0}, {1, 1}}");
slider7.contentVerticalAlignment = UIControlContentVerticalAlignmentCenter;
slider7.continuous = YES;
slider7.enabled = YES;
slider7.hidden = NO;
slider7.highlighted = NO;
slider7.maximumValue = 1.000;
slider7.minimumValue = 0.000;
slider7.multipleTouchEnabled = NO;
slider7.opaque = NO;
slider7.selected = NO;
slider7.tag = 0;
slider7.userInteractionEnabled = YES;
slider7.value = 0.500;

UILabel *label5 = [[UILabel alloc] initWithFrame:CGRectMake(478.0, 364.0, 163.0
label5.frame = CGRectMake(478.0, 364.0, 163.0, 21.0);
label5.adjustsFontSizeToFitWidth = YES;
label5.alpha = 1.000;
label5.autoresizesSubviews = YES;
label5.autoresizingMask = UIViewAutoresizingFlexibleRightMargin | UIViewAutores
label5.baselineAdjustment = UIBaselineAdjustmentAlignCenters;
label5.clearsContextBeforeDrawing = YES;
label5.clipsToBounds = YES;
label5.contentMode = UIViewContentModeLeft;
label5.contentStretch = CGRectFromString(@"{{0, 0}, {1, 1}}");
label5.enabled = YES;
label5.font = [UIFont fontWithName:@"Helvetica" size:17.000];
label5.hidden = NO;
label5.lineBreakMode = UILineBreakModeTailTruncation;
label5.minimumFontSize = 10.000;
```

```
label5.multipleTouchEnabled = NO;
label5.numberOfLines = 1;
label5.opaque = NO;
label5.shadowOffset = CGSizeMake(0.0, -1.0);
label5.tag = 0;
label5.text = @"Label";
label5.textAlignment = UITextAlignmentLeft;
label5.textColor = [UIColor colorWithRed:0.000 green:0.000 blue:0.000 alpha:1.0
label5.userInteractionEnabled = NO;

[view2 addSubview:imageview4];
[view2 addSubview:label5];
[view2 addSubview:segmentedcontrol6];
[view2 addSubview:slider7];
[view2 addSubview:activityindicatorview8];
```

This makes the hierarchy part at the end to be much more readable and easier to understand. This was something that Rudi's script was doing after the code was produced, and I wanted to integrate that functionality inside the tool.

Enjoy!

More nib2objc fun

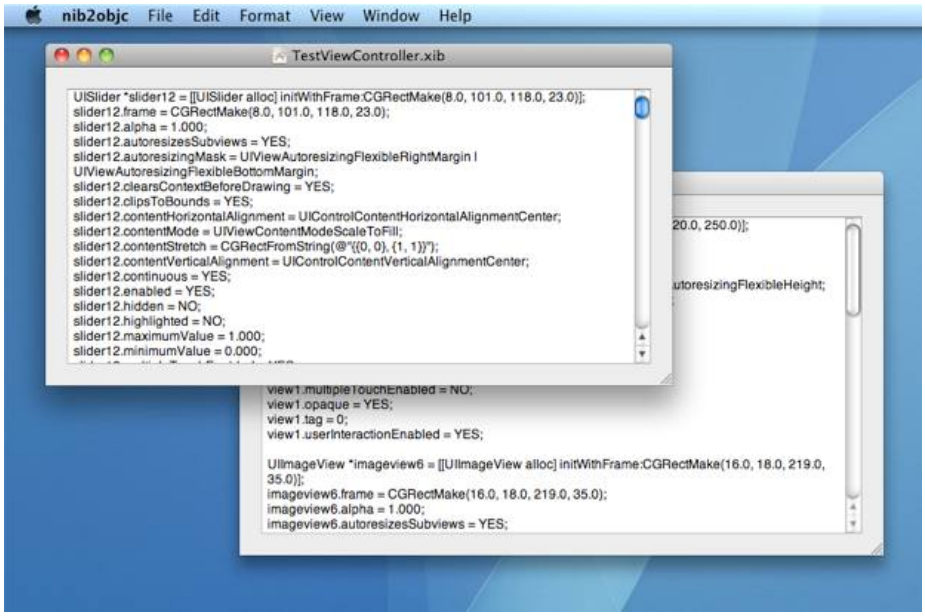
Adrian Kosmaczewski

2010-07-18

There are many things that I wanted to do with nib2objc[1] since day one, but of course sometimes you just don't have the time to implement them all. But here go two new additions to the project, that will make nib2objc even simpler and more fun to use: a Mac application and a Mac OS X Service! The command line version is OK, but again, not everyone feels at home with command line tools.

Mac OS X Application

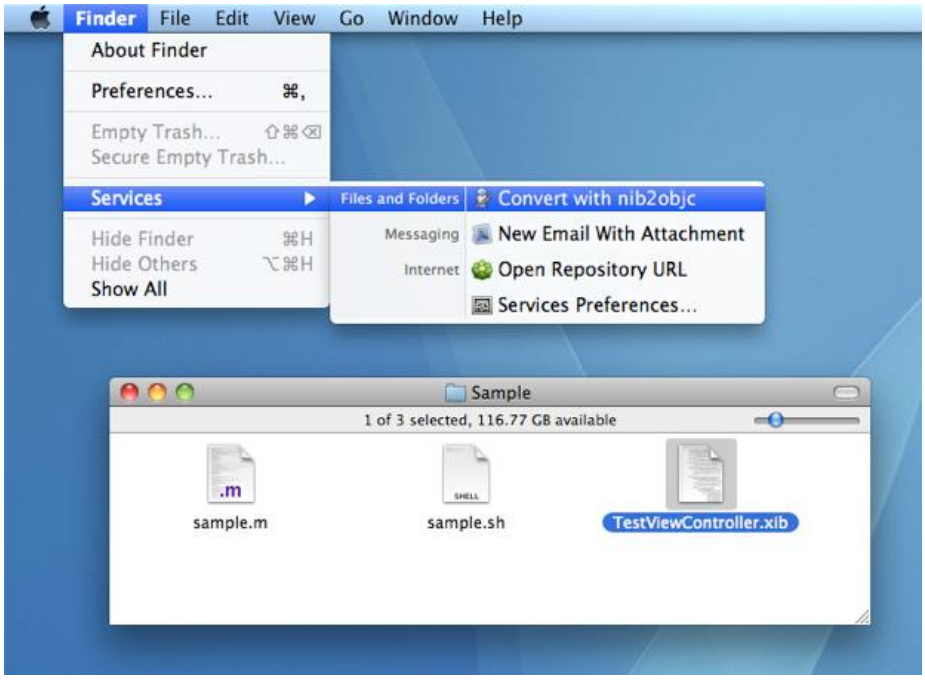
I've crafted a quick and dirty Mac application, available in the repository (under the "GUI" folder) that can be used to open several NIB files and see their contents in separate windows. Simple and useful. You can drag and drop xib files from the Finder to the dock icon and open any file in your projects.



[1] http://github.com/akosma/nib2objc

Mac OS X Service

It cannot be simpler than this: select a file in Finder, open the “Services” menu, select the “Convert with nib2objc” entry, and your clipboard will contain the full source code of your NIB file. You only have to paste it into your preferred code editor!



I hope you will enjoy using these tools, and feel free to send your ideas and patches, as usual, via Github[2].

[2] http://github.com/akosma/nib2objc

Welcome to the Company!

Adrian Kosmaczewski

2010-07-22

Many people have asked me why, when I was an employee, I used to change jobs so often. The answer stands in between my own curiosity to take on new challenges, and the various assholes I had to deal with through the ages. Just as an example of this last case, here goes a true story, one that stands between being a candidate story for The Daily WTF[1], or as sample material for The No Asshole Rule[2] book by Bob Sutton. You decide.

Prologue

A couple of years ago I found a job as a PHP + JavaScript developer in a small company in Geneva, Switzerland. I remember going to their offices two or three times, and having several interviews with various people there; one of them was the lead PHP developer of the company, the other being the CEO, a relatively well-known person in the tech area in Geneva; both shall remain nameless. The last interview I had was with the CTO, who would be my direct boss, as I was told.

They finally chose me, and very happily I signed the contract. I handed my resignation for my current job at the time, but had a couple of months of work to do before leaving (this is usual practice in Switzerland, one that I despise deeply, but that you are legally forced to follow). All in all, three months passed between me signing the contract and the first day of my new job.

The First Day

So one day, I headed to Geneva to start my new job. I arrive at around 9am to the address where the interviews had taken place, and, oh surprise… there was nothing. Stay with me: **there was nothing**. Not a sign in the wall indicating that the company used to be there, not a single desk, not a phone plugged on the wall. Nothing.

[1] http://thedailywtf.com/

[2] http://www.amazon.com/Asshole-Rule-Civilized-Workplace-ebook/dp/B000OT8GV2%3FSubscriptionId%3D0F0YTN83N46JSX6KDT02%26tag%3Dakosmasoftwar-20%26linkCode%3Dxm2%26camp%3D2025%26creative%3D165953%26creativeASIN%3DB000OT8GV2

Puzzled (to say the least), I asked the first person I met in the hallway about the company, and she told me that they had left a couple of months ago. I asked if she knew where they went, but she told me that she did not.

I was really, really worried by now. Had I signed a contract with some kind of fake company that had just left the country to the Bahamas or Luxemburg? I called their phone number. The automated voice at the other end told me that the number was not in service.

Oh dear.

After what must have been like 60 minutes of going back and forth in the hallways asking for some kind of information about the company, one guy told me that they had moved not far from there. Finally a clue! He even gave me an address, so I left as quickly as I could. I was one hour late to my new job; you do not do that in Switzerland.

On my way, I could not help thinking things like, “why wouldn’t they call me to tell me that they moved to a new place? What’s going on?”

So around 1030am I arrived to this new address, got into the building, and checked in at a reception desk that was standing there. I asked the names of the people that had interviewed me. The guy told me that nobody with that name worked there. Then I asked, “I’m looking for the company such and such”, and he told me that no, this was a private bank (there are lots of them in Geneva), so I must have been given the wrong address.

Bummer. Back to step one.

The guy, nevertheless, told me one interesting thing; in the warehouses behind the bank there was this new “startup center” with brand new offices, and the company might as well be there. I thanked the guy, and started investigating the area.

The word “investigating” is the correct one. It felt like being Columbo looking for a murderer.

Indeed, behind the bank there was a huge, new complex with many new offices and small companies popping up. The building was an old factory that the city of Geneva had bought a couple of years before, and where you could rent cheap office space. Perfect for startups. But in the main entrance, there was no sign of the one I was looking for.

It was almost 11am, and I was about to give up. My cell phone had not rang, nobody called from their office; my wife told me that nobody had called home either. If they were around, they really did not care about me.

Just when I was about to leave the building, I asked one guy cleaning the hallway about the company name. He told me that he had never heard about it, but that there was a huge sign near the entrance of the parking with company names, and that given that the building was

fairly new, not all company names had been set up in every entrance of the building, so I might as well check that one out.

I left the building, went to the entrance of the parking, and finally! I saw the name of my new employer. Together with the indication of how to get there by foot: a 10 minute walk from where I was. I said to myself, well, what the heck. Let's go.

When I crossed the door, I saw yet another reception desk, this time with a huge sign behind the receptionist with the name of the company. This was finally the good one. That was at around 1115am; I had been touring Geneva for over 2 hours looking for this company by now.

I tell the girl that I am starting today my new job, and she tells me that it was her first day too, so she did not know anyone, so she guided me to an office marked "Human Resources", which looked quite appropriate for the occasion.

"Welcome to the company!"

The HR guy gets up from his desk, greets me and tells me that he started 2 weeks ago and that he did not know that I was starting that day, but that was OK, welcome to the company anyway! He guides me to the sector of the open space where the technical team works, and I finally see some familiar faces, together with some 30 people I had not seen three months ago.

The company had had an explosive growth in just 3 months.

Anyway, they point me to a crappy chair and table in the open space and they called that a "desk", and I said, OK, let's score some more points. Even better, the IT manager comes up to me and says "oh sorry, I don't have a computer for you, I didn't know you were coming today".

I sit down, awkwardly, as everyone resumed their tasks in an awkward silence, a mix of "I don't know anyone here" and "I hope it's 5pm soon". Probably one of the worst feelings I have had in a work environment in a while.

Meet the boss

30 minutes later (it was almost noon, and I was really starving by now), while I was sitting on my chair without having anything to do or anyone to talk to, a guy looking like a hawaiian surfer comes up to me and tells me that he was my boss. Which was strange, because he was not the CTO I mentioned earlier, but given that everything had changed so much, I was not surprised.

The surfer takes me to a meeting office, we sit down for what I think it is going to be my first work meeting, and he tells me that he has

been appointed to this boss role last month, that they are dropping PHP altogether, and that they will be doing the new system using Java. The guy tells me that he knows that I despise Java[3] (he read it on this very blog, actually) and that he does not like me not liking Java[4]. But he cannot fire me, because he has not hired me, so my role is undefined and, as a matter of fact, I have nothing to do there.

It was 12am, and by now I know I will not be doing long in this company.

To make a long story short, a few days after that I went to the office of the CEO, I gave them my resignation letter, and they just told me, literally, “OK, bye”.

That was it.

Epilogue

After two years, I was told that the Java system was never finished. The company still exists but has completely changed its business model, and the CEO has left the country and moved his company with him.

I also learnt that the original PHP developer, one of the guys who interviewed me, the one who worked his ass off for 4 years building the only system that was actually bringing cash, was also being dismissed from the new team because he was not a Java guy. He was let go a couple of months after I left. Nobody cared that he actually knew how the original system worked, how the business worked, or that he gave 4 years of his work for a company that greeted him with another “OK, bye”.

My path to independence started that very day. Dealing with that kind of crap (of which I have many more nice anecdotes that I will write about very soon) is what tells me that I do not want to be an employee again. I prefer to starve rather than being treated like shit.

PS: you will not find the company name in my LinkedIn profile[5], for reasons that should be obvious by now.

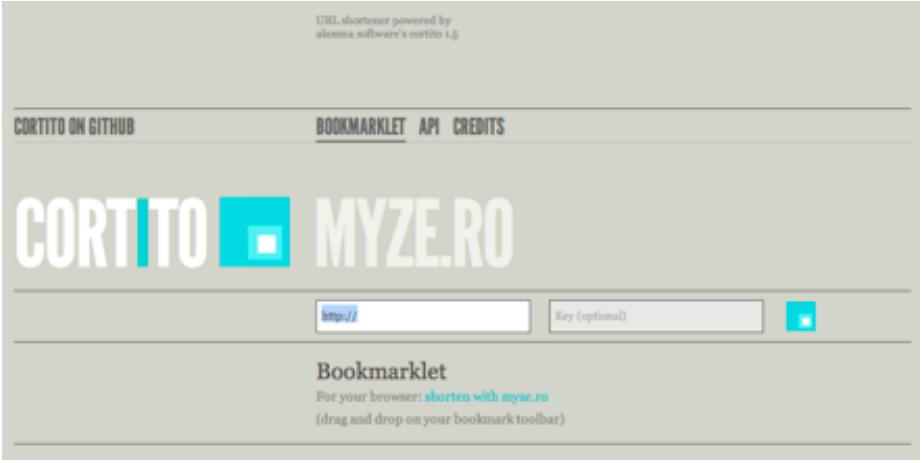
[3] /blog/not-exactly-what-i-meant/
[4] /blog/to-java-or-not-to-java/
[5] http://www.linkedin.com/in/akosma

A nicer cortito, courtesy of Zerofee

Adrian Kosmaczewski

2010-07-22

Those who follow me on Twitter[1] are certainly aware of my endless sequence of shared links, some of them shortened using the akos.ma[2] domain, itself powered by cortito[3], one of my most popular open source projects.



Well, Ela and Paul of Zerofee[4] asked me to give them a hand to install an instance of cortito into their own shortening domain, myze.ro[5], and in return they contributed an awesome design for cortito!

[1] http://twitter.com/akosma
[2] http://akos.ma/
[3] http://github.com/akosma/cortito
[4] http://zerofee.org/
[5] http://myze.ro/

Disclaimer: Ela and Paul are my cousins :) And if you have not heard about Zerofee, check their website[6], their blog[7] and their Twitter streams: @Zerofee[8] and @zerofee_ela[9].

Finally, for those who do not know it, cortito is written using Ruby on Rails[10], and I have been told that it is even used internally by AT&T Interactive[11]. Feel free to check the code out and use it in your own domain!

[6] http://zerofee.org/
[7] http://zerofee.org/goodthinking/
[8] http://twitter.com/Zerofee/
[9] http://twitter.com/zerofee_ela/
[10] http://rubyonrails.org/
[11] http://attinteractive.com/

Migrating iPhone 3.x apps to iPad and iOS 4.0

Adrian Kosmaczewski

2010-07-26

Right now, creating Universal Applications[1] for the iPod touch, the iPhone and the iPad is not really a straightforward task. The current panorama of iOS-compatible software and hardware platforms is getting more and more complex, and this blog post is a small guide (by no means exhaustive) of tips and tricks that have helped me get my apps running in as many platforms as possible, with as few headaches as possible.

First of all, a few warnings:

- I assume your current applications run in iPhone OS 3.1 as a minimum requirement. Apple does not accept any more applications targeting the 2.x frameworks these days, and the developer tools do not include those anymore.
- I assume your projects compile and link without warnings[2]; many API calls have been deprecated between 2.x and 4.0, so you'd better have code using the latest methods and no deprecated ones.

Panorama

Waiting for an unification of the iPad and the iOS 4 frameworks (probably later this year[3]), the current panorama of software and hardware iOS platforms looks as follows:

	iPhone			iPod touch		iPad
	3G	3GS	4	2nd gen	3rd gen	
iPhone OS 3.1	x	x		x	x	
iPhone OS 3.2						x
iOS 4.0		(ls)	x	(ls)	x	
Year Released	2008	2009	2010	2008	2009	2010
RAM (MB)	128	256	512	128	256	256

[1] http://devimages.apple.com/iphone/resources/introductiontouniversalapps.pdf

[2] /blog/objective-c-compiler-warnings/

[3] http://adage.com/digital/article?article_id=144670

	iPhone			iPod touch		iPad
CPU (MHz)	620	833	1 GHz	620	833	1 GHz

(ls): limited support, particularly for multitasking.

For practical purposes, I've omitted in the table above the fact that the iPhone 3G is (theoretically) able to run iOS 4. Something that has been empirically proved to be a bad idea[4].

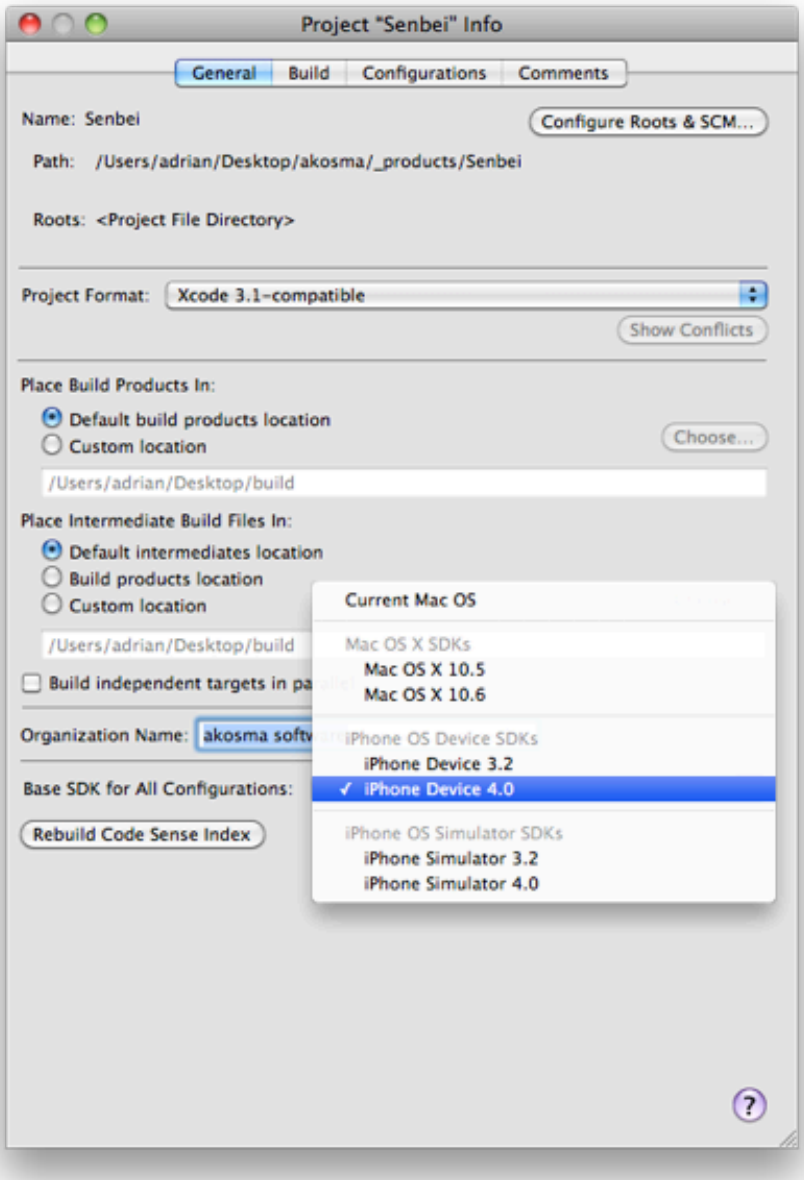
Upgrading Xcode Projects to iOS 4

Your Xcode 3.1 iPhone project can be migrated to iOS 4 as follows:

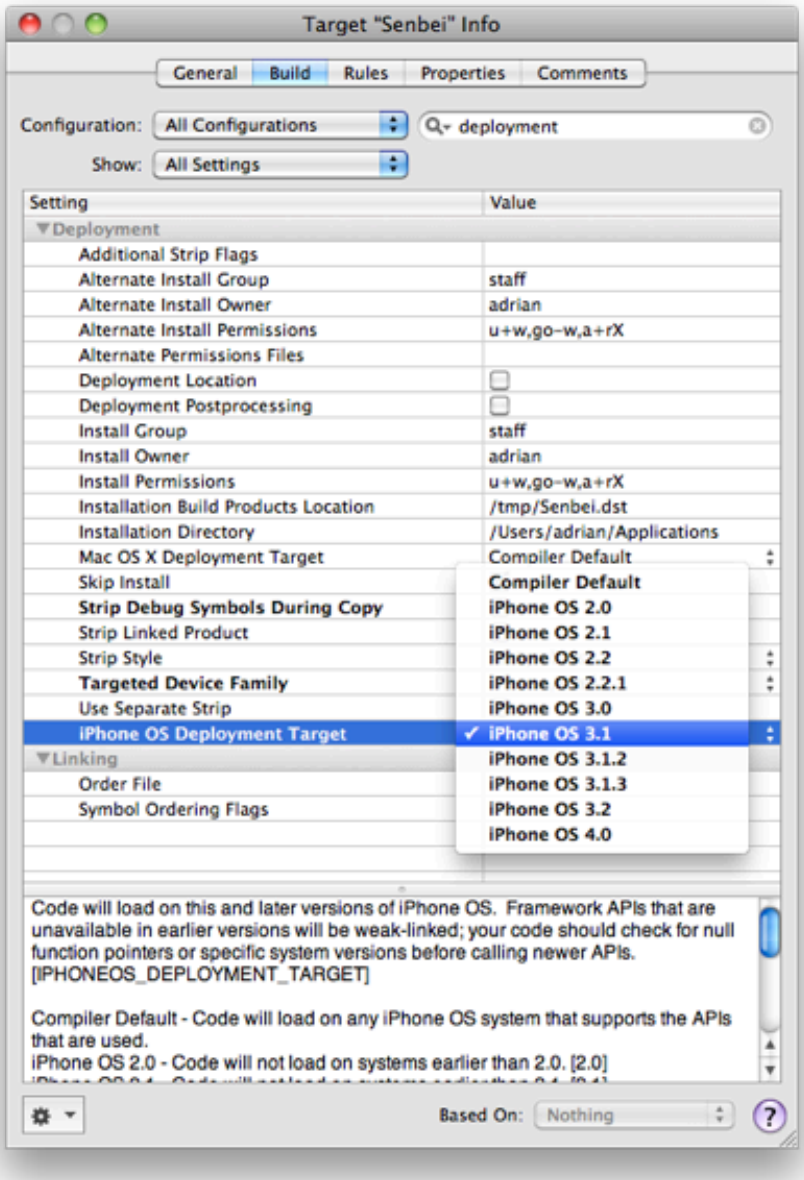
- If you are using an SCM with branching support, create a new branch for all of the operations below; you don't want to impact your "trunk", or main branch with all these changes, as more urgent bugs might appear while you are doing this.
- Download the latest SDK from Apple[5]. This is required by Apple, actually. And the latest version of Xcode (3.2.3 at the time of this writing) allows you to write applications compatible with all the platforms in the table above.
- Open your project with Xcode 3.2.3; select the "Project / Edit Project Settings..." menu and change the "Base SDK value to iPhone Device 4.0 (yes, even if your project is meant to target iPhone OS 3.1... stay with me!)

[4] http://www.tuaw.com/2010/07/22/ios-4-and-iphone-3g-is-a-match-made-in-whats-the-opposite-of/

[5] http://developer.apple.com/iphone/

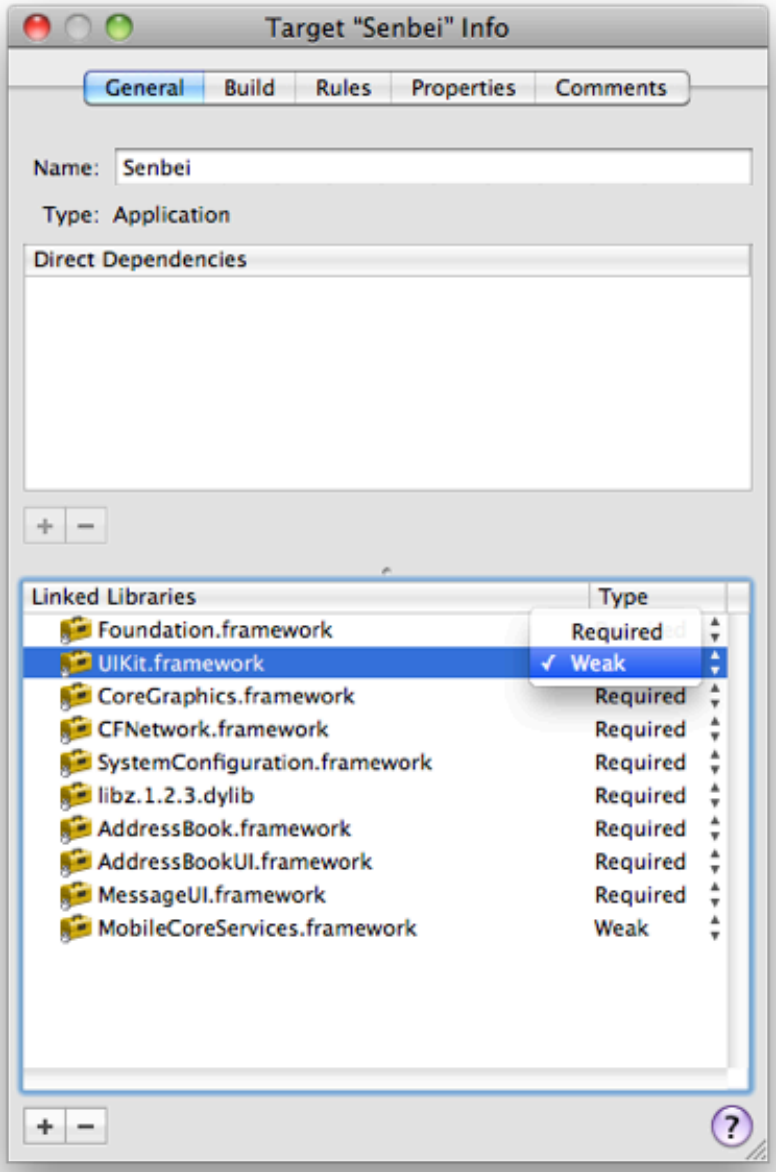


- In the “Targets” group of your Xcode project, open the “Info” panel, select the “Build” tab and change the “iPhone OS Deployment Target” entry for all the configurations of your target to “iPhone OS 3.1”. This will allow your code to be loaded in earlier versions of iOS, even if your application uses iOS 4 as a base SDK for all configurations:



- Set the UIKit framework, as well as any new framework provided by iOS 4, to link as “Weak”; for example, in the screenshot below, taken from the settings of my open source Senbei application[6], you can see that UIKit is weakly linked, as well as the MobileCoreServices framework (which does not exist in iPhone OS 3). This prevents the application from crashing at startup in versions of iOS earlier than 4, because newer versions of UIKit bring new classes and APIs that were not available before:

[6] http://github.com/akosma/Senbei



- Change your Entitlements.plist file, because iOS 4 applications submitted to the App Store require a new format in the Entitlements.plist file. In most cases, you can just delete the previous file, and create a new one from Xcode ("File / New File...", then select "Code Signing / Entitlements" in the dialog that appears).

Old Entitlements.plist:

```
<plist version="1.0">
```

```
<dict>
    <key>get-task-allow</key>
    <false></false>
</dict>
</plist>
```

New Entitlements.plist:

```
<plist version="1.0">
<dict>
    <key>application-identifier</key>
    <string>$(AppIdentifierPrefix)$(CFBundleIdentifier)</string>
    <key>keychain-access-groups</key>
    <array>
        <string>$(AppIdentifierPrefix)$(CFBundleIdentifier)</string>
    </array>
</dict>
</plist>
```

- Update your icons and graphics for the new “Retina” display of the iPhone 4; Apple has published a special guide[7] that provides all the information required in terms of sizes, formats and naming conventions.

After doing the steps above, you should have an application running on iPhone OS 3.1, 3.2 (as an iPhone application) and 4.0, offering exactly the same functionality and ready to be sent to the App Store.

Create a Universal iPhone and iPad application

Creating an iPad-compatible application from your iPhone app requires much more than just adapting your source code; the user

[7] http://developer.apple.com/iphone/library/qa/qa2010/qa1686.html

experience of iPad apps is a completely different beast than that of iPhone apps, so I strongly recommend you ask a graphic and/or UX designer for help[8]. I can't stress this too much; iPad apps aren't just "big" iPhone apps. They are much, much more.

Technically speaking, in the simplest of terms, you should be able to migrate most UIKit-based application following the standard path provided by Apple:

- Select your target in Xcode and choose the "Project / Upgrade Current Target for iPad..." menu.
- Use some compile-time and runtime tricks to get different parts of your code to execute in different environments, as explained by Jeff LaMarche[9]:

```
#if __IPHONE_OS_VERSION_MAX_ALLOWED >= 30200
    if (UI_USER_INTERFACE_IDIOM() == UIUserInterfaceIdiomPad)
        NSLog(@"iPad Idiom");
    else
#else
        NSLog(@"iPhone Idiom");
#endif
```

Testing your code in different devices

My advice is the following: don't upgrade your old iPhone to iOS 4, particularly if you own a 3G. Keep it running under 3.1 and get the new iPhone 4 as soon as you can. This has two major advantages:

- You will be sure that there aren't crashes or unexpected situations in iPhone OS 3.1, particularly when you are using new APIs in your code.
- The iPhone 3G and 3GS are slower than the new iPhone 4, but they are still (probably not for long) the most widely deployed, particularly in those countries where the iPhone 4 has not yet been released. This will help you create code that uses low memory, and that runs faster in newer models.

In my personal situation, I have my old iPhone 3G running iPhone OS 3.1.3, and a second generation iPod touch running iOS 4. And of course, my iPad runs iPhone OS 3.2.1.

Adapting your code for different platforms

Although Apple recommends using the UI_USER_INTERFACE_IDIOM macro[10], sometimes you need a tighter control in your code. For this, I have used three different techniques:

[8] http://www.smashingmagazine.com/2010/04/16/design-tips-for-your-ipad-app/

[9] http://iphonedevelopment.blogspot.com/2010/04/few-more-notes-on-creating-universal.html

[10] http://developer.apple.com/iphone/library/documentation/General/Conceptual/iPadProgrammingGuide/StartingYourProject/StartingYourProject.html

- Use a different class as the app delegate in your iPad application; instead of making your code a mess of “if” and “else” statements or preprocessor routines, make your iPad application create an instance of a different class, one that might or might not be a child class of the existing application delegate. This way, you can completely separate the code of both platforms, and each can load a different UI control hierarchy, without impacting the existing iPhone OS 3.1 or iOS 4.0 infrastructure. You can specify a different class for your iPad app delegate directly in your MainWindow-iPad.xib file, created for you when you selected the “Project / Upgrade Current Target for iPad...” menu (one more advantage of using Interface Builder files in your projects!):

- Check for symbols at runtime, taking advantage of the dynamic nature of Objective-C; for example, the code below only executes in iOS 4 or later, because the ADBannerView is not available in previous versions of the iPhone OS. This code will run without problems in iPhone OS 3.1! You can also use NSObject’s “respondToSelector:” method to perform runtime checks of the capabilities of your current platform, which might help you branch your code in different directions depending on the context:

```
- (void)showAdvertising
{
    Class klass = NSClassFromString(@"ADBannerView");
    if (klass)
    {
        ADBannerView *adView = [[ADBannerView alloc] initWithFrame:CGRectMake(0
        adView.currentContentSizeIdentifier = ADBannerContentSizeIdentifier320×
        adView.delegate = self;
        [self.view addSubview:adView];
    }
}
```

- Include and use in your project the UIDevice extensions by Erica Sadun[11]. This very handy set of categories provides a wealth of options, making your code much more readable and easier to understand. Most importantly, it is actively maintained by Erica, and this means that future versions of the categories will be able to detect more features of iOS.

Finally, MPMoviePlayerController requires special attention, because it's widely used in many applications and also because it has been greatly improved (and changed) between iPhone OS 3.1, iPhone OS 3.2 and iOS 4. Those changes include, in some cases, deprecated APIs, so here I'll just link to this excellent article by John Muchow[12] in the iPhone Developer Tips blog, which explains all the problems and the possible solutions.

Conclusion

As you could see, creating universal apps is not easy, but it isn't impossible either; it requires a bit of attention and lots of testing. Do you have any tips or links that you would like to share? Feel free to add more information in the comments below.

[11] http://github.com/erica/uidevice-extension

[12] http://iphonedevelopertips.com/video/getting-mpmovieplayercontroller-to-cooperate-with-ios4-3-2-ipad-and-earlier-versions-of-iphone-sdk.html

JAOO 2010

Adrian Kosmaczewski

2010-08-27

It's that time of the year again: JAOO 2010[1] is getting closer, with sessions and workshops from Sunday October 3rd to Friday 8th. JAOO has been held since 1996, and it is a multi-language, non-vendor focused, eclectic, flexible conference where you can learn about those technologies you've never heard about, or those you never had the time to explore in depth. I can't but recommend to any software engineer to attend, given the breadth and the quality of the presentations.

This year's speaker lineup is awesome as usual; just a few names to get you excited: Rob Pike (creator of the Go programming language[2], at Google), Martin Odersky (creator of Scala[3], from the EPFL[4]), Douglas Crockford (the legendary JavaScript guru[5] and the creator of JSON[6], from Yahoo!), Scott Chacon (founder of Github[7]), Jeff Sutherland (father of the Scrum[8] methodology), James Gosling[9] (creator of Java), Jim Coplien[10] (C++ legend), Martin Fowler[11] (no need to say more), Tim Bray[12] (father of XML)… and much more!

And akosma software is proud and humbled to announce that I will be there to talk about, you guessed it, iOS, the iPhone and the iPad. I am going to host two sessions[13] this year:

[1] http://jaoo.dk/
[2] http://golang.org/
[3] http://www.scala-lang.org/
[4] http://www.epfl.ch/
[5] http://www.crockford.com/javascript/javascript.html
[6] http://www.json.org/
[7] http://github.com/
[8] http://www.scrumalliance.org/
[9] http://en.wikipedia.org/wiki/James_Gosling
[10] http://en.wikipedia.org/wiki/Jim_Coplien
[11] http://martinfowler.com/
[12] http://en.wikipedia.org/wiki/Tim_Bray
[13] http://jaoo.dk/aarhus-2010/speaker/Adrian+Kosmaczewski

- On Sunday, October 3rd, from 9 AM to 12 PM, the “Write your first iPhone App” training, where I will guide you into learning the tools and paradigms required to create a small yet functional iPhone application.
- On Monday, October 4th, from 1:30 PM to 2:30 PM I will present my findings on “Accessing Web Services From iPhone and iPad Applications”, diving into a subject of interest to many developers new to the platform.

In any case, if you are following me on Twitter[14] (which you should, definitely) you can benefit of a 20% discount on the ticket price by using the “JAOOspeakerfollower” promotion code! I will be there the whole week, so don’t hesitate to stop me to say hi; I’d love to meet you in person there, and to share a good conversation about software engineering, trends and buzzwords!

[14] http://twitter.com/akosma

Kuroshio Sea

Adrian Kosmaczewski

2010-08-30

This is probably one of the most mysterious and beautiful videos on the web. Enjoy.

Kuroshio Sea - 2nd largest aquarium tank in the world - (song is Please don’t go by Barcelona) from Jon Rawlinson on Vimeo.

Danske Bank iPhone Application

Adrian Kosmaczewski

2010-09-21

akosma software is proud and honored to announce a joint collaboration with Trifork Switzerland GmbH[1] and Trifork A/S[2] in the development of the Danske Bank iPhone Application[3].

[4]

The Danske Bank[5] is one of the largest banking organizations in Scandinavia. This iPhone application is one of the first of its kind, not only featuring a gorgeous and innovative design, but also offering a full suite of mobile banking services, including:

- Currency exchange
- Instant mobile payments
- Live account status information
- Quick access to contact info
- GPS- and map-enabled ATM and branch search

In just one week, the application has gathered more than 200 positive customer ratings, with an average of 4 stars and a half! More than 35'000 downloads in a couple of days[6] are the proof that the future

[1] http://trifork.ch/
[2] http://www.trifork.com/
[3] http://itunes.apple.com/dk/app/danske-mobilbank/id388423462?mt=8
[4] http://itunes.apple.com/dk/app/danske-mobilbank/id388423462?mt=8
[5] http://danskebank.com/
[6] http://twitter.com/jlatrifork/status/25019516557

of banking is mobile; users want to be able to transfer money as fast, as secure and as easily as possible.

The application is available for free worldwide, but of course just a subset of functionality will be available for users without a Danske Bank account.

[7]

[7] http://itunes.apple.com/dk/app/danske-mobilbank/id388423462?mt=8

Mnesia iPhone Application

Adrian Kosmaczewski

2010-09-22

akosma software is proud (again) to announce another partnership to create a new application: this time is Mnesia[1], a cool iPhone application developed jointly with Trifork Switzerland GmbH[2] and Keith Bingman[3], an extraordinary designer from Zurich!

[4]

What is Mnesia? Mnesia is a gorgeous (Retina display enabled) application that helps you remember important things and put a structure to your life or work.

To quote Gerald Weinberg:

> What you don't know may not hurt you, but what you don't remember always does.

Mnesia let you work with checklists in a structured way. First you design a checklist in an easy way on your iPhone. An example could be a "Go Camping" checklist where you add each task you need to perform before you are ready to go camping.

When you have designed the checklist you can start using it by just completing a task in the list. Doing that Mnesia takes a copy of the

[1] http://trifork.ch/products/mnesia/
[2] http://trifork.ch/
[3] http://twitter.com/kbingman
[4] http://itunes.apple.com/ch/app/mnesia/id392269920?mt=8

checklist and stores it in a folder named “Go Camping”. Each time you go camping a new copy is placed in that folder.

[5]

Checklists in Mnesia can be organized in folders and subfolders by simply “drag and dropping” them, similarly to the iOS Springboard application. You can share a checklist by simply forwarding the checklist as an email or you can use the iTunes file share option.

Features:

- Create check list templates
- Order check lists in folders
- Use check lists
- Keep track on when a check list was used
- Keep track on how much work that has been performed according to the check list
- Share check list templates by sending them by email or iTunes file share

With Mnesia you will always have the checklists in the palm of your hand, exactly where and when you need them.

You can download Mnesia from the App Store[6]! It is a free, ad-supported application. Enjoy!

[5] http://itunes.apple.com/ch/app/mnesia/id392269920?mt=8
[6] http://itunes.apple.com/ch/app/mnesia/id392269920?mt=8

Dezeen Watchstore Web App

Adrian Kosmaczewski

2010-09-23

Another day of announcements at akosma software: this time, a fruitful and fun collaboration with Zerofee[1] to create a dynamic visual identity for the Dezeen Watchstore[2], and their Dezeen Watchstore iPhone & iPad Web Application[3] (best viewed with Safari on iOS, iPhone & iPad!).

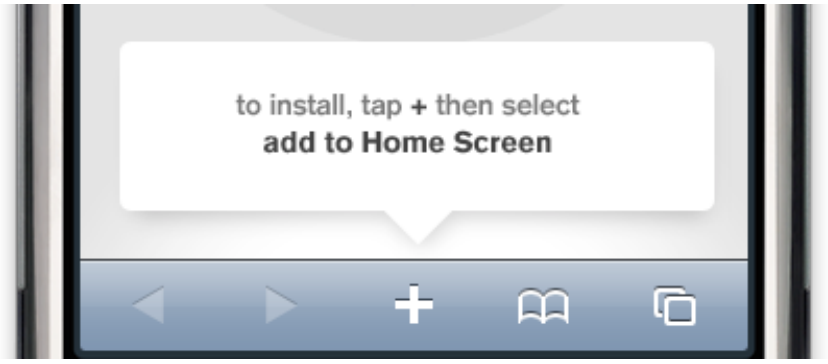
Dezeen[4] is an online magazine, one of the most popular and influential architecture and design blogs on the Internet. It has been included in Time magazine's Design 100 list of the most influential forces in global design[5], and in Design Week magazine's Hot 50[6] list of key figures in design.

[1] http://zerofee.org/
[2] http://dezeenwatchstore.com/
[3] http://dezeenwatchstore.com/clock/
[4] http://www.dezeen.com/
[5] http://www.dezeen.com/2008/05/15/dezeen-in-time-magazines-design-100/
[6] http://www.dezeen.com/2008/03/02/design-week-hot-50/

Their online watch store will be launched soon, and in the meantime the Zerofee team came up with a brilliant idea: a timeless, innovative, non-traditional clock, built using JavaScript, CSS, and the Raphaël JavaScript library[7].

The clock (explained in the figure below) features different circles, representing the current month, hour, minute and second of the day. The circle representing the hours does a complete circle ever 24 hours, instead of the typical 12 hours required. The circle representing the seconds grows and disappears every minute.

One of the greatest advantages of iOS web applications is that you can bypass the App Store review process altogether! So feel free to point your iOS browser to http://dezeenwatchstore.com/clock/ and install the application in your home screen. Enjoy and share it with your friends!



[7] http://raphaeljs.com/

Random Thoughts on Partnerships

Adrian Kosmaczewski

2010-09-26

A couple of months ago I had a very interesting conversation with a friend of mine, who happens to be a close business partner in many different ventures. During this conversation, one of his phrases, probably the simplest of all, struck me and stayed in my mind:

“Business is about giving and receiving”.

Now, don’t get me started on that chapter of “Friends”, where Joey writes a speech[1] to celebrate Chandler and Monica’s wedding, and all he can come up with is a series of “giving and having and sharing and receiving” phrases. Stay with me; I will try to elaborate on this point.

Being a constant learner in the business area, I take these phrases as precious information to shape the destiny of my own business. akosma software[2] is my first company, and to my own amazement, my wife and I are able to live out of it; said like this, it might sound ridiculous. But it is a huge statement for me, an incredible source of pride. It is part of my very personal way of watching the world and ourselves.

Most people leave their employee status behind to start a company with the inner feeling of being the next Bill Gates or, these days, the next Steve Jobs. The idea, particularly in the tech business, is to become insanely rich. Much more than you really need to live a decent life. And to screw all those who block your path, and yes, why not, build an empire down the road.

Bullshit.

Let’s go back to basics: the basic idea of business, the one that lays at the bottom end of any human activity which involves the exchange of goods and services, is to be able to get what you need for a living, in exchange of your work or its product. Remember? That’s how it all started around 10 thousand years ago. And the basic ideas behind akosma software lie in very simple terms:

- To be able to live out of the product of my work, offering my own work, services and ideas;

[1] https://www.imdb.com/title/tt0108778/quotes/?item=qt0410137&ref_=ext_shr_lnk

[2] http://akosma.com/

- To be able to build something sustainable, that eventually might feed other people too;
- To learn new things, and to meet interesting people during the trip.

Most businesses have completely lost their touch with reality, not only because of the kind of products offered, but simply because of their way to perform their work. They treat people in such a way[3] (inside and outside of their organizations) in such a way that it makes Jason Fried's speech sound like revolutionary, instead of what it really is: good old, down-to-earth, common sense[4].

When Jason says that most companies do their best to get out of business, he's absolutely right; simply because most people have forgotten why they go to work every day. That's why books like Getting Real[5] or REWORK[6] are successful. That's why Bob Sutton[7]'s books are successful. That's why Tim Ferriss' "4 Hour Workweek"[8] became a bestseller.

Businesses, companies, managers, employees, we have all lost touch with reality.

Running a business is not just about how much money you can make; taking care of that is important, but not enough. It's not about trying to screw people just for the sake of making a few more bucks, all while having nice words about social responsibility in your website. It's not about filling your mouth with the ecology word while not even recycling paper in your printer room.

Don't get me wrong; I consider myself a liberal. I believe in the free market and I resent governments sneaking into our lives. I believe in being able to associate and create new things respectful of what surrounds us as a society and an environment[9]. I also believe that cooperatives are the best type of organization for small businesses, but I do not believe in democratic project management processes. I believe in vision and direction for products and projects[10], and I believe in a redistribution of the earnings (and the losses) among all those who helped create something new. I believe in small teams, committed to (not just involved in) their projects.

Most importantly, I want to believe in the people I work with. Let me repeat that: I work **with** people; they do not work **for** me, I do not work **for** them. I believe in us, whoever is "us" in that group.

---

[3] /blog/welcome-to-the-company/

[4] http://www.inc.com/magazine/20091101/the-way-i-work-jason-fried-of-37signals.html

[5] http://gettingreal.37signals.com/

[6] http://37signals.com/rework/

[7] http://bobsutton.typepad.com/

[8] http://www.fourhourworkweek.com/blog/

[9] /blog/reducing-the-carbon-footprint/

[10] /blog/saving-a-failing-project/

Business is about not being hypocrite. Business is about being up-front about what you do, about the value that you add, about being passionate about your craft, about being proud of your work and your colleagues. Business is about saying no at the right time, and about never saying yes without care and attention.

Business **is** about giving, sharing and receiving. A good friend of mine[11] told me once that I should never say "thanks" to him; a simple "wow, cool" was better. Being able to marvel oneself with whatever comes is a key element in happiness, the second key element being giving back to others whatever happiness we have.

When you focus business on people, long term partnerships invariably become strong friendships.

[11] http://sandorastarita.blogspot.com/

JAOO Workshop: Building your first iPhone Application

Adrian Kosmaczewski

2010-10-03

This morning I've had the immense pleasure of guiding many students in Århus, Denmark, in taking their first steps with iOS, the iPhone, Xcode and Interface Builder. This event was part of the workshops proposed during the JAOO conference[1], and which target a very wide range of topics.

I've published the slides of my presentation in my SlideShare account[2]; you can also download the projects we created together[3] during this event.

Build your first iPhone app[4]

View more presentations[5] from Adrian Kosmaczewski[6].

As a followup, I recommend those interested in continuing their path towards creating iOS application to read this article appeared today in Mobile Orchard[7].

I hope all the participants enjoyed the event and I look forward to see your apps in the App Store soon!

[1] http://jaoo.dk/aarhus-2010/

[2] http://www.slideshare.net/akosma

[3] Projects.zip

[4] http://www.slideshare.net/akosma/build-your-first-iphone-app

[5] http://www.slideshare.net/

[6] http://www.slideshare.net/akosma

[7] http://mobileorchard.com/how-to-make-iphone-apps-part-1-xcode-suite-and-objective-c-3/

Swissair

Adrian Kosmaczewski

2010-10-03

When I was a student in university, I used to work in Geneva Airport, aka GVA, as a part-time luggage handling employee, an “auxiliaire” as we were called, in a now extinct company once called Swissair.

The job consisted mostly of waiting for the airplanes to park near the gate, open the cargo bays, offload whatever there was inside them, and reload them with more luggage, cargo boxes and mail bags. After that, we would close the cargo bays and stay clear of the engine ranges until the plane left the gate. Rinse and repeat. That was my routine, 4 hours a day, 3 to 5 days a week, from August 1995 until December 1997.

Let’s be frank; as a whole, this job sucked big time, the pay wasn’t very good (20 Swiss Francs per hour, 25 after 10pm and on Sundays; I let you do the math), and there were many health risks. Then how on Earth could I stand for so long the blithering cold of Swiss winters, the endless humming of the reactors, the kerosene-filled atmosphere, or the killing summer sun reflected on the concrete of the tarmac, for hours and hours?

There were some good reasons:

- Because I loved aeronautics, and for me, being close to the landing gear of a Jumbo Jet or an Airbus A-400 was an out-of-this-planet experience. I still remember the first time I saw such a beast approaching.
- Because you could renegotiate your working hours every month with great flexibility, which is perfect when you are studying.
- Because it was an endless source of both funny and tragic anecdotes, many of which I keep telling to my friends to this day, and which would fill a whole blog by themselves.
- And, last but not least, because you could travel all over the world, wherever Swissair went (that was an awful lot of places back then, including my beloved Buenos Aires) with a **tremendous discount**.

And I mean tremendous; I could have a return ticket GVA - EZE in Economy class for around 180 Swiss Francs. You read correctly; after a certain amount of hours (300, if I remember well, which meant between 4 to 6 months of work in my case), you would get a spe-

cial employee voucher, of which you could trade one for a discounted European ticket, and two of them for a transcontinental return flight.

The catch (there's always a catch) was that your ticket was of the "absolute-low-priority-you-might-not-fly-at-all-you've-been-warned" kind, and there was always the risk of not getting into the plane because it was full; thankfully, that never happened to me. But, to compensate, if Economy was full, you could get promoted to Business class (which actually did happen to me once: a great flight from Buenos Aires to Zurich in Business class, never to be forgotten!)

All in all, I went 4 times to Buenos Aires in one year, for less than what you pay for a single flight to the same place. All while getting a relatively decent living every month (for a cheap student at least). Not bad, and definitely worth the hassle of smog, rain, snow, heat, pains in the back, and overall organizational chaos that reigned in Swissair back then.

Because the thing is that, internally, Swissair was a complete mess.

Actually I was not surprised when I learnt that the company had gone the way of the dodo a few years after I left (and no, don't tell me it was a because of the cheap tickets to Argentina!). During my time there, the airline industry was undergoing a tremendous amount of change, and still, the management was pathetically convinced that people would still pay premium tickets for an EasyJet-level quality of service (Swissair tickets were extremely expensive back then). Middle and top managers started getting into the rotten habit of paying themselves huge bonuses while at the same time downsizing the staff. Employees' salaries and benefits were gradually reduced and suppressed, security measures were skipped, everyone was treated like shit.

The SAirGroup came and went. Many reorganizations made the headlines. In Crossair (Swissair's little sister company, serving short-range destinations), pilots were loudly complaining of earning around 3000 Swiss Francs per month (a relatively low salary by any standard in Switzerland), when some of their Swissair peers were getting much more, up to 6 times (!) that amount, basically doing the same job. One day, Swissair moved most of its intercontinental flights from Geneva to Zurich, and the direction of the GVA airport sold most of its Swissair stock right after that decision was taken. It was during that time that EasyJet came into the scene, and benefited enormously of all this mayhem.

For those who remember, Swissair during the '90s was more a source of national embarrassment rather than pride. And to top it off, the catastrophe of the JFK-GVA flight 111 in 1998[1] did not help.

In our luggage handling service, which later was spun off into a company (IMHO clumsily) named "Swissport" (French-speaking workers

[1] http://en.wikipedia.org/wiki/Swissair_Flight_111

would rename it to the French equivalent of “Swiss pork”), the uncannily shortsighted view of the management saw hiring more and more “auxiliaires” (like myself) as the perfect way to increase shareholder value (or some other MBA crap like that); lots of my friends did work there at the time, but the truth is, we never cared about a company that saw us (and treated us) as a commodity to be replaced at any time. As soon as we could, we took off (pun intended) for a nicer workplace or, like me, for another country altogether (my last flight on a Swissair aircraft was in January 1998, when I returned to live in Argentina; I never used the return ticket then).

Their long term employees were seen as a liability, instead of an asset. Everybody resented that.

In the end, it looked to us as if lots of powerful people wanted to destroy the company on purpose, an explanation that, even if looking like a conspiracy theory, is believed by many of my old colleagues. Since leaving I met several of them, some of which used to do this job full time; many have quite a few health problems now, the lesser of which are recurrent back or joint pains; frankly, just by breathing that shitty airport smog you could get very ill, very quickly, and that’s just the top of the iceberg.

Nobody is really happy about what happened to Swissair, but we all agree in one thing: all of this could have been avoided with just a bit of common sense, a better treatment to long term employees, and wiser leadership. The problem is, some Swiss companies just simply don’t know what common sense means, even less how to adapt to changing market conditions.

I’m writing this sitting in the waiting room of Gate D85 at Amsterdam airport, seeing the KLM employees unloading and loading stuff from those blue planes, and I remember a time when I used to do that job, too.

I think I’ll start pouring some souvenirs about those times in the months to come.

Integrating iOS Applications with Backend REST Services

Adrian Kosmaczewski

2010-10-04

A couple of hours ago I finished my presentation at JAOO[1], a discussion of what I've learnt about integrating REST services in iOS apps while creating the iPhoneWebServicesClient project at Github[2].

This project showcases different transport formats and libraries to consume web services from an iPhone application. It features a server application written in PHP, a command-line script that saves the output of the server application in different formats, and an iPhone client application (compatible with iPhone OS 3.0 and higher). The server application reads a MySQL database and outputs data in the following formats: HTML, JSON, YAML, XML (arbitrary format), SOAP, Property list (Binary and XML), CSV, and Protocol Buffers.

As promised, here go the slides and of course, check the project in Github[3] which actually contains the real thing!

Although not everyone liked it[4] (you definitely can't please everyone) I've got good feedback from many other attendees and all in all I'm happy to have shared what I've learnt in this project. Feel free to fork the code and add support for other libraries and configurations! I'd love to receive feedback about this project.

[1] http://jaoo.dk/aarhus-2010/presentation/Accessing%20Web%20Services%20From%20iPhone%20and%20iPad%20Applications

[2] http://github.com/akosma/iPhoneWebServicesClient

[3] http://github.com/akosma/iPhoneWebServicesClient

[4] http://www.version2.dk/artikel/16464-jaoo-den-foerste-roede-seddel

Soft-Shake 2010

Adrian Kosmaczewski

2010-10-05

Voici l'annonce officielle de la conference Soft-Shake 2010:

> Agilité, iPhone et Java voilà les trois thèmes de la conférence Soft-Shake qui se tiendra pour la toute première fois le 18 octobre 2010 à Genève. La conférence est organisée en commun par les associations de l'Agile Tour de Genève, le Groupe des Développeurs iPhone de Suisse romande et le Geneva JUG.
>
> Les échanges entre des communautés n'ont pas pour habitude de se rencontrer et notre idée a été de les regrouper autour d'un événement, avec des sessions pour chacun. Nous avons rajouté un track incubateur qui permettra de tester des sujets qui mériteront peut-être un track spécifique l'année prochaine (NoSQL, Cloud Computing).
>
> Côté iPhone, nous aurons des sessions sur le développement piloté par les tests, de façon Agile, ou encore comment distribuer des applications iPhone/iPad en entreprise. Allez voir sur le site web les autres sessions, notamment en Java et Agilité.
>
> Chose importante: le prix est raisonnable. 100 CHF (~ 80 €) pour une journée de conférences et le repas de midi inclus.
>
> Toutes les sessions et l'inscription sont sur le site internet de l'association Soft-Shake:
> www.soft-shake.ch[1]
>
> Pierre-Yves Bertholon
> Membre organisateur de Soft-Shake 2010.

[1] http://www.soft-shake.ch

QCon London 2011: Call for Papers in the Mobile Track

Adrian Kosmaczewski

2010-10-08

I am honored to announce that I will be taking part in QCon London 2011[1], but this time not as a speaker, but as host of the mobile track!

QCon is a series of conferences taking place in San Francisco[2] and London every year. It is a joint venture between Trifork[3] and InfoQ[4], and as explained in the QCon website[5],

> The QCon Conferences are organized by the community, for the community. As software developers and architects ourselves, we wanted to craft the ultimate conference that we would find outstanding, as attendees. The result is a high quality conference experience where a tremendous amount of attention and investment has gone into having the best content on the most important topics presented by the leaders in our community, staged in an intimate environment with the best food and comfort needed to support as much learning and networking as possible.

And as a proof, check out the tremendous amount of information exposed during the 2010 edition[6]!

We are currently working towards the 2011 edition of the London event, taking input from you, to get the best speakers and the best content. As part of this effort, I am officially opening a channel for those interested to submit ideas for presentations and sessions in the mobile track of the next edition of QCon. We are interested in sessions about mobile technologies, namely:

[1] http://qconlondon.com/
[2] http://qconsf.com/
[3] http://www.trifork.com/
[4] http://www.infoq.com/
[5] http://qcon.infoq.com/
[6] http://www.infoq.com/articles/qconlondon-2010-summary

- iOS
- Android
- Web technologies
- … and more!

So, if you want to participate as a speaker about any of those technologies, just use the contact form in this site[7]! I look forward to hearing from you. Feel free to pass this information to whoever you think might be a good candidate, too!

[7] /about/

How knowing C and C++ can help you write better iPhone apps, part 1

Adrian Kosmaczewski

2010-10-11

While learning how to write iOS applications, you will often encounter the phrase "learn C first"[1]. Writers of Cocoa applications apparently benefit from knowing about C (sometimes even C++), but it is not very clear to many new developers how this actually works.

The obvious question being "why should I learn C if actually I have to use Objective-C to create my apps?", and the answer "because Objective-C is a superset of C" is not really useful, albeit technically correct.

This series of two articles will provide some pointers (no pun intended) to answer that question, in the hopes that new iOS developers will get a copy of the K&R[2] and the Stroustrup[3] books soon.

Objective-C

Before starting with this article, I would like to remind my readers about some key facts of Objective-C, often overlooked by tutorials and teachers, and which explains much of the characteristics of Cocoa as well:

1. All methods (at instance and / or class level) are public, virtual and overridable. You can have @public, @private and @protected instance variables (also known as "ivars"), but methods are public. And yes, you can override class methods in subclasses, too, which is not a very common pattern (you can do that in C++ through templates, though). Polymorphism is the

[1] http://stackoverflow.com/questions/180549/learn-c-first-before-learning-objective-c/180832#180832

[2] http://www.amazon.com/Programming-Language-2nd-Brian-Kernighan/dp/0131103628%3FSubscriptionId%3D0F0YTN83N46JSX6KDT02%26tag%3Dakosmasoftwar-20%26linkCode%3Dxm2%26camp%3D2025%26creative%3D165953%26creativeASIN%3D0131103628

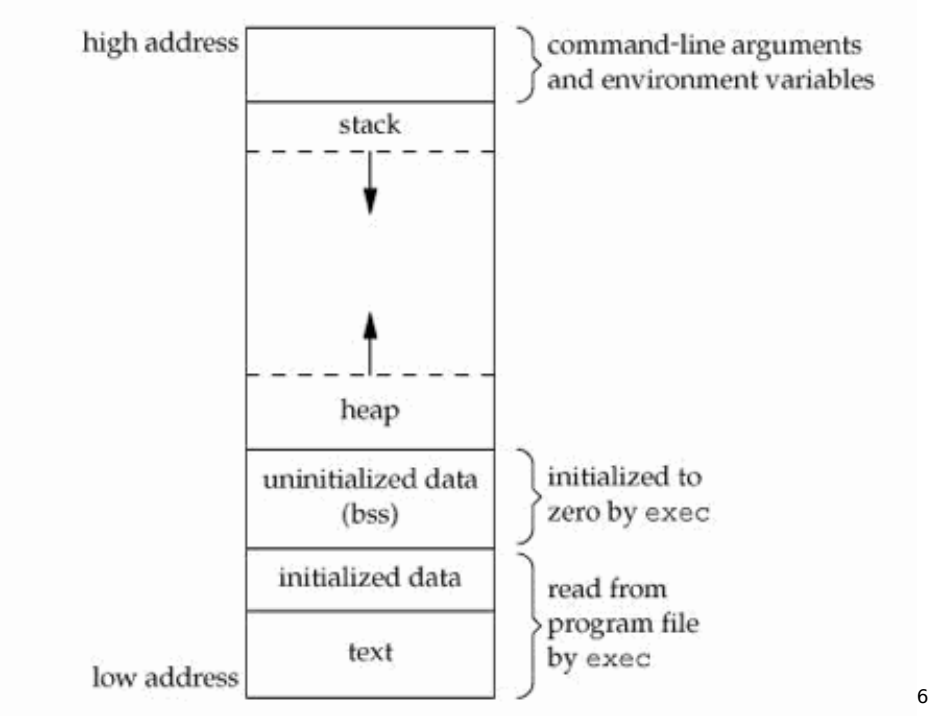
[3] http://www.amazon.com/C-Programming-Language-Special/dp/0201700735%3FSubscriptionId%3D0F0YTN83N46JSX6KDT02%26tag%3Dakosmasoftwar-20%26linkCode%3Dxm2%26camp%3D2025%26creative%3D165953%26creativeASIN%3D0201700735

key to understanding Objective-C.[4]

2. There is no formal concept of abstract classes. You can override the alloc and init methods of some class to avoid creating instances from it, but a priori, you can create an instance from almost any class.
3. You can have as many "root classes" as you want. You are not forced to inherit from NSObject, although in most cases that is what you do. In Cocoa there are two root classes already defined for developers to use: NSObject and NSProxy.
4. There is no concept of namespaces. That explains the prefix crazyness all over the place, because all classes live in the same planet, so you should better use a nice prefix for your stuff. In my case I usually publish my classes with the "AKO" prefix.

Stack and Heap

This is the probably the most important thing to know about iOS programming. Many developers, particularly those coming from Java, .NET, PHP or other garbage-collected environments, just don't know that the computer memory used by applications is not a uniform space, and that running code uses space in three different, yet complementary, regions of memory, or as computer scientists refer to them, "segments": the text, the stack and the heap.[5]



[6]

[4] http://en.wikipedia.org/wiki/Type_polymorphism
[5] http://www.ualberta.ca/CNS/RESEARCH/LinuxClusters/mem.html
[6] http://www.boundscheck.com/knowledge-base/c-cpp/memory-layout-in-c/342/

The “text segment” is where the application code lives in memory while your application runs. Every instruction is there, every single function and procedure and method and executable code lives in that part of memory until your application exits. Usually you don’t really have to know anything about the text segment, other that it is there.

When the application starts, the main() function is called and some space is allocated in the “stack”. This is another segment of memory allocated for your application, and that’s where the memory required for the variables of your functions are allocated. Each time you call a function in your program, a section of the stack called a “frame” is allocated in the stack. The local variables of this new function are allocated there.

[7]

As the name implies, the stack is a “last in & first out” (or LIFO) structure, and when functions call other functions, stack frames are created; when those functions exit, their frames are destroyed automatically (that’s why stack objects are usually referred to as “automatic” objects). In your code, an “end” statement or a closing curly brackets “}” visually indicate a delimiter of stack context; when the current execution goes beyond that element, you can be sure that the stack frame is gone.

The stack region is at the origin of the “stack overflow” concept, which frequently happens when using recursive functions; you allocate too many frames, and you run out of space in the stack. Boom. Another interesting problem associated with stacks is that of “buffer overflows”, which constitutes one of the most common security problems caused by badly written C code. When you write memory contents in the stack, and you don’t check the size of your structures, you might end up writing something on the text segment... which basically means that you might as well be rewriting the code of the application, as it runs. Bad, very bad things can happen then.

[7] http://www.futuregov.asia/photologue/photo/2008/aug/30/stack-papers/

 [8]

Finally, the “heap” (also called the “data” segment) provides a storage medium that can last throughout the execution of functions; global and static variables live on the heap, until the application exits. To access whatever data you have created in the heap, you require at least one “pointer” on the stack, because that’s how your CPU can access data in the heap; through the stack. You can think of a pointer as just an integer variable that holds the number of a particular memory address in the heap (it is actually a bit more complicated, but that’s the basic picture).

To summarize: the CPU uses the value of the pointer in the stack to access or, as they say, “dereference”, the structure in the heap. Without pointer on the stack, no access to the object on the heap. And here lies the reason of all memory leaks: if you lose your pointer for whatever reason, then you have lost track of an object in the heap. And given that the heap is cleared only when your application exits, you have “leaked” memory, and you will never be able to recover it again (unless you restart your program, that is, but we don’t want to ask your user to quit and restart to solve that problem, do we?).

In C++ you can create objects in both the stack and the heap, and to know how to differentiate between both regions, you have to pay attention to the syntax used to create them:

```
// The variable does not use a "star"!
// Created on the stack, don't use "new"
SomeClass stackObject;

// do not call delete stackObject!!!

// Pay attention to the "star"...
// it's a pointer on the stack!
// We are initializing it to NULL,
// as a rule of thumb.
SomeClass *heapObject = NULL;

// And to create the object, use "new"
// to have it created on the heap
heapObject = new SomeClass;

// you must call this to avoid a leak!
delete heapObject;
```

In the case of the stack object, its destructor is called automatically when the stack frame is removed. In the case of the heap object, what

[8] http://linguiniontheceiling.blogspot.com/2008/10/thats-madame-trash-heap-to-you.html

goes away is the pointer variable in the stack... but not the object on the heap! To avoid that problem, in C you have to balance every malloc() (or calloc() or similar function) with a free(). In C++ you have to balance every call to "new" with a call to `delete`.

Now, imagine for a second that you have an object in the stack (an automatic object) that points to some other object in the heap, and that our first object is prepared to call the destructor of the second one. This is how the auto_ptr[9] (or "smart pointer") class of the standard C++ template library (STL) works; it takes advantage of the fact that variables on the stack are automatic. Instead of having to manage manually the lifecycle of your objects, an `auto_ptr` object on the stack will call delete automatically when it goes out of scope:

```
#include <iostream>
#include <memory>
using namespace std;

int main(int argc, char **argv)
{
    // created on the heap
    SomeClass *heapObject = new SomeClass;

    // now the stackPointer owns the heapObject
    auto_ptr<SomeClass> autoObj(heapObject);

    // Print non-NULL address of heapObject
    cout << *autoObj << endl;
}
```

How does all of this apply to Objective-C? Well, it turns out that the current versions of Objective-C only allow us to create objects on the heap. Apparently creating objects on the stack was possible in early versions of Objective-C, but that isn't the case anymore. All objects that you manipulate in Objective-C live in the heap[10] and as such, if you lose the pointer to them, you have a memory leak. Here goes some code in Objective-C with an obvious memory leak:

```
- (void)doSomething
{
    NSObject *obj = [[NSObject alloc] init];

    // do something else with obj...

    // and now we forget to release, and boom:
    // Memory leak! This is because the
    // "obj" pointer in the stack is lost
```

[9] http://en.wikipedia.org/wiki/Auto_ptr

[10] Well, that is not entirely true; there is one kind of Objective-C object that lives in the stack, and that is blocks[11]. But this is an exception, one that I will tackle in a separate blog post.

```
    // when this stack frame is removed...
    // ... but the heap object stays!
}
```

What can you do to avoid that? Two things:

1. Remember how in C you balance malloc() with free, and how you balance "new" with "delete" in C++? Well, in Objective-C you have to balance every call to "alloc", "copy" or "retain" with a "release".
2. You can also use the "autorelease" method, to add your object to the current autorelease pool, so that it will be automatically disposed of in the next iteration of the run loop cycle. OK, this is slightly more complex, but basically this also solves the memory leak problem.

Here goes some code illustrating the two solutions in detail:

```
// First solution
- (void)doSomething
{
    NSObject *obj = [[NSObject alloc] init];

    // do something else with obj...

    [obj release];
}

// Second solution
- (void)doSomething
{
    NSObject *obj = nil;
    obj = [[[NSObject alloc] init] autorelease];

    // do something else with obj...

    // Don't release obj! The object referenced
    // by that pointer will be disposed
    // in the next run loop iteration.
}
```

However, please, please, avoid the following code, which stretches the use of the autorelease pool to the level of abuse[12]:

```
for (NSData *data in veryLargeArrayWithImages)
{
    UIImage *image = [UIImage imagedWithData:data];
    // do something with image
}
```

because this will create an unnecessary large number of instances that will remain in memory until the next run loop cycle (which will

[12] Autorelease pools are definitely material for another blog post.

probably happen after the end of the current loop, that is, in a long CPU time in the future). Instead, either use a temporary autorelease pool, or use the non-autoreleased version of the code above (which I prefer, particularly in the iPhone):

```
for (NSData *data in veryLargeArrayWithImages)
{
    UIImage *image = [[UIImage alloc] initWithData:data];
    // do something with image
    [image release];
}
```

All of these memory management techniques are explained in detail in my article 10 iPhone Memory Management Tips[13].

Mike Ash has written an extensive article about stack and heap objects[14] that will help you understand all of these issues even better. Finally, to see visually how the stack and the heap collaborate with each other, and how to keep an object-oriented mindset when writing code in C, I recommend watching on iTunesU the excellent Programming Paradigm[15] course by professor Jerry Cain from Stanford University.

Object ownership

Ownership is a key concept in C++ memory management: whenever an object creates or copies another, the first one becomes the "owner" of the second. And as such, it is the task of the first one to destroy the second. Here goes some code that shows that pattern; here some class requires another one, and the ownership relationship is enforced throughout the lifetime of both objects.

```
// Forward declaration of some owned class
class Owned;

// Declaration of the owner
class OwnerClass : BaseClass
{
public:
    OwnerClass();
    virtual ~OwnerClass();
    const Owned& getOwnedObject();

private:
    Owned* _owned;
};
```

[13] /blog/10-iphone-memory-management-tips/

[14] http://www.mikeash.com/pyblog/friday-qa-2010-01-15-stack-and-heap-objects-in-objective-c.html

[15] http://itunes.apple.com/WebObjects/MZStore.woa/wa/viewPodcast?id=384233005

```
// Constructor implementation
OwnerClass::OwnerClass()
: BaseClass()
, _owned(new Owned)
{
}

// Destructor implementation
OwnerClass::~OwnerClass()
{
    delete _owned;
}

// Provide to clients a reference
// to the underlying owned object
const Owned& OwnerClass::getOwnedObject()
{
    return _owned;
}
```

In Objective-C, the chain of ownership is important to solve the problem of circular references; this is a common problem when creating delegation relationships between objects. When an object is delegate of another, the reference between the object and its delegate must be weak:

```
// A delegate protocol declaration
@protocol OwnedDelegate <nsobject>

@optional
- (void)ownedObjectSaysSomething:(Owned *)owned;

@end

// Interface of the owner
@interface OwnerClass : NSObject <OwnedDelegate>
{
@private
    Owned *_owned;
}

@property (nonatomic, retain) Owned *owned;

@end

// Interface of the owned
@interface Owned : NSObject
{
@private
    id<OwnedDelegate> _delegate;
}
```

```
@property (nonatomic, assign) id<OwnedDelegate> delegate;

@end
```

Pay attention at the @property declarations in the code above. The OwnerClass “retains” the Owned object, but the Owned class just “assigns” the delegate. This way, we know clearly who’s responsible of releasing who: it is the OwnerClass dealloc method that will do that, and not the other way around!

```
@implementation OwnerClass

@synthesize owned = _owned;

- (id)init
{
    if (self = [super init])
    {
        _owned = [[Owned alloc] init];
        _owned.delegate = self;
    }
    return self;
}

- (void)dealloc
{
    _owned.delegate = nil;
    [_owned release];
    _owned = nil;

    [super dealloc];
}

#pragma mark -
#pragma mark OwnedDelegate protocol

- (void)ownedObjectSaysSomething:(Owned *)owned
{
    // do something now
}

@end
```

If you don’t follow this pattern, you might end up with a circular reference, and the reference counting scheme of Objective-C won’t be able to solve this problem, which means that you might end up leaking two instances at once!

free() vs. delete vs. dealloc

In C, free() is a function taking any pointer as parameter, and whose purpose is to mark the region of memory pointed by the parameter as available for the application. It is up to the application developer to pay attention to properly clean any owned objects in the structures stored in that region, as the standard C runtime makes no assumptions whatsoever about those secondary relations.

In C++, delete is an operator, and its task is to ultimately call the destructor of the target object, starting by the class of the current object, until it reaches the last class in the inheritance chain, and it does this immediately when called, in a synchronous fashion. Your constructors do not have to call the constructor of the base class in their implementation; it is, however, strongly recommended to mark your destructors as virtual[16], to ensure that the delete operator does this in all cases.

On the other hand, dealloc in Objective-C is a polymorphic, virtual, overridable method, not an operator neither a function, and it is never called directly by your code. It is called in a kind of asynchronous fashion, when the current run loop ends, and when the retain count of the current object reaches zero. This means that you might not know exactly when that happens, and actually, you shouldn't care. The important thing is to release all the things you were holding on to (see the "object ownership" section above).

Another important fact of dealloc is that, being polymorphic, only the implementation of the current class will be called, so you must always remember to add a call to [super dealloc] at the end of your own dealloc. And always, always add it at the end, not at the beginning[17], of your own dealloc.

Conclusion

I hope this information will be useful to developers new to the iOS platform. Objective-C and Cocoa share many characteristics with other similar platforms and frameworks written in C and C++, and developers working with garbage-collected runtimes are not used to know the layout of structures in memory. I think, however, that knowing this is useful also in such environments.

In the second part of this series I will dive into variable initialization, bitwise masks, functions and other related aspects of C and C++ programming, and how they relate to Objective-C.

Acknowledgement: Many thanks to Ariel Rodríguez[18] for reviewing early drafts of this post.

[16] http://www.parashift.com/c++-faq-lite/virtual-functions.html#faq-20.7

[17] http://stackoverflow.com/questions/909856/why-do-i-have-to-call-super-dealloc-last-and-not-first/909925#909925

[18] http://volonbolon.net/

Update, 2010-10-13: Ariel has written an excellent follow up to this article, explaining the problem of stack overflows[19].

[19] http://volonbolon.net/post/1305559150/stack-overflow

Article dans com.in Magazine

Adrian Kosmaczewski

2010-10-12

> Le 7 septembre dernier, à Zurich, s'est tenu "Mobile App Stores", un colloque à propos des plus importants systèmes de distribution de software pour téléphones mobiles. Au programme, des présentations de l'App Store de Apple, de l'App World de BlackBerry, l'Ovi Store de Nokia et également sur des applications web mobiles, construites avec les derniers standards tels que HTML5 et CSS3. Le ton a été donné d'emblée. Marc Carrette et Rami Omari de Sogeti (du groupe Cap Gemini) ont clairement mis en relief l'impact économique qu'ont eu les App Stores depuis 2005. A ce jour, plus de 250'000 applications y sont enregistrées. Ce n'est donc pas un hasard si Apple considère son App Store comme "le plus important événement dans l'histoire du software mobile."

Article paru dans le numéro d'Octobre 2010 de com.in Magazine[1] (télécharger le PDF[2])

[1] http://www.cominmag.ch/
[2] /press/Cominmag2010.pdf

Le Temps Sortir Dîner Universal Application

Adrian Kosmaczewski

2010-10-13

We are proud to announce a new iOS application: this time is Sortir Dîner[1], an elegant and useful iOS universal application for the Swiss newspaper Le Temps[2], working on the iPhone, the iPad and the iPod touch, developed jointly by akosma software[3] and Trifork Switzerland GmbH[4], and featuring a gorgeous, exclusive design by the Geneva-based Bontron agency[5].

[6]

Sortir Dîner is a premium restaurant guide for Switzerland (available in French only at the moment) providing the description and directions of the finest restaurants around. The contents of the application are taken from the guide “Le Petit Suisse à Table”, and features over 400 restaurants in Switzerland.

From the editorial[7] in the edition of today of the newspaper Le Temps:

[1] http://www.letemps.ch/tout_le_temps/applications/sortir_diner/

[2] http://www.letemps.ch/

[3] http://akosma.com/

[4] http://trifork.ch/

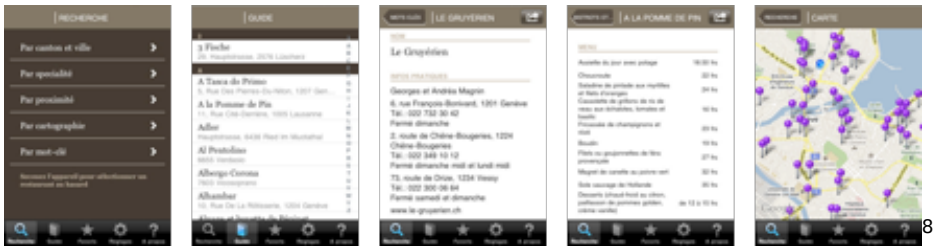
[5] http://www.bontron.ch

[6] http://ax.itunes.apple.com/ch/app/id385167598?mt=8#

[7] http://www.letemps.ch/Page/Uuid/c4c06300-d641-11df-9552-bf32c651740c/Sortir_Dîner_lapplication

> La Suisse recèle des trésors gastronomiques souvent méconnus. Sortir Dîner est le nouveau moyen efficace de les atteindre. Cette application pour iPhone et iPad, proposée par Le Temps, donne accès à une version électronique du guide de référence Le Petit Suisse à table. Quatre cents tables en Suisse romande, en Suisse alémanique et au Tessin y sont répertoriées. La recherche d'un restaurant est aisée: elle se fait par le nom de l'établissement, par le canton et la ville, par spécialité, par proximité, par cartographie ou encore par mot clé. Développée en collaboration avec les Editions Texto, Sortir Dîner est téléchargeable sur iTunes/AppStore ou sur le site letemps.ch/application. D'une valeur de 12,50 francs, Sortir Dîner est mise gracieusement à disposition des abonnés du Temps qui sont invités, au moment du téléchargement, à utiliser leur nom d'utilisateur et leur mot de passe.

A quick showcase of some application features:

 [8]

- Find restaurants classified by canton and city.
- Browse restaurants classified by speciality.
- Filter restaurants specialized in fish, having a terrace, or only opening on Sundays.
- Search restaurants using free text keywords.
- Find restaurants around your current location (using the built-in GPS.)
- Each restaurant features a custom screen, with the menu, contact information, the possibility to share it with friends via e-mail, and many other options.
- Keep bookmarks to your favorite restaurants in the "Favoris" screen.
- Shake the device to discover a random restaurant (only available in the iPhone and iPod touch.)
- Browse the map of Switzerland and discover new restaurants.
- On the iPad, use gestures to browse any list of restaurants: swipe to the left to see the next restaurant, swipe to the right to see the previous one.
- Suggest new restaurants to the editorial team, for inclusion in the next versions of the application!

Sortir Dîner is compatible with iPhone OS 3.1 and above, and with iPad

[8] http://ax.itunes.apple.com/ch/app/id385167598?mt=8#

3.2 and later. It has been tested on the current iPads models, 2nd and 3rd generation iPod touches, and on iPhone 3G, 3GS and 4 devices.

You can download Sortir Dîner from the App Store[9]! It is a free application for subscribers to the Swiss newspaper Le Temps (ad-supported), but other users can buy a non-restricted, ad-free access to the information for CHF 12.50. Enjoy!

[9] http://ax.itunes.apple.com/ch/app/id385167598?mt=8#

Migrantes de Vicente López

Adrian Kosmaczewski

2010-10-23

Me acuerdo de una fiesta en lo de Viole (una de tantas, qué se yo), allá por el ’90, en plena época del Bilingüe (época del Austral, de la hiperinflación, del penal de Codesal, y de otras alegrías). Si no recuerdo mal, la casa de Viole quedaba cerca de Las Heras y Chacabuco[1], o una esquina así en Florida, a unas pocas cuadras de la avenida Maipú.

Me acuerdo que esa noche estábamos todos los del Bilingüe, más amigos y hermanos; creo que fue un sábado por la noche. De la gente que estuvo ahí de esa noche, varios nos fuimos para seguir nuestras vidas en otros lugares.

- Carolina se fue de Florida (Partido de Vicente López) a Florida (USA), y labura en el ramo inmobiliario[2].
- Fernando, el hermano de Viole, es actor[3] en Nueva York, USA.
- Santiago es profesor[4] en California.
- Me dijeron que Magalí se fue a vivir a Pittsburgh, USA[5].
- Rosana se fue a vivir al Uruguay.
- Andrés es ingeniero agrónomo[6] en Entre Ríos.
- Lucía estuvo en Francia unos años, pero después volvió a Argentina.
- Déborah también se fue a vivir a Francia.
- Leticia estuvo unos años en Brasil, pero no tengo noticias de ella desde hace años.
- Eugenia, creo yo, esta en España (?).
- Y yo me vine a Suiza, dos veces, pero eso ya lo saben.

20 años no es nada.

[1] http://maps.google.com/maps?f=q&source=s_q&hl=en&geocode=&sll=37.0625,-95.677068&sspn=59.467068,93.427734&ie=UTF8&hq=&ll=-34.533677,-58.483368&spn=0.007672,0.011405&z=17

[2] http://activerain.com/carolinah7

[3] http://www.imdb.com/name/nm2564527/

[4] http://www2.haas.berkeley.edu/Faculty/oliveros_santiago.aspx

[5] https://www.dobsearch.com/people-finder/view.php?t=1287864041&sessid=2c998123089294d2f1118d2ca2ffd8b3&searchnum=57119228183

[6] http://208.86.249.19/~copaer/index.php?grupo=2&seccion=5&nota=76

Talking at the EPFL Today

Adrian Kosmaczewski

2010-11-11

Today at 1400 CET Adrian will be talking at the world-famous Ecole Polytechnique Fédérale de Lausanne[1] (EPFL), giving an introduction to iOS development to students.

[2]

The video of the presentation will be available as live streaming on http://video.epfl.ch[3], so be sure to check it out!

[1] http://epfl.ch/
[2] http://video.epfl.ch/
[3] http://video.epfl.ch/

swissinfo.ch iPad application

Adrian Kosmaczewski

2010-12-27

We are thrilled and extremely proud to announce a new iPad application: this time is swissinfo.ch[1], an innovative news reader application for swissinfo[2], the heir of the now extinct Swiss Radio International which ceased broadcasting back in 1999. swissinfo.ch offers an international and Swiss perspective on major news around the world, in 9 languages, updated every day with exclusive content. swissinfo.ch is a branch of the SRG, the state-owned media conglomerate. This iPad app was developed by akosma software, and was released to the public in December 11, 2010.

The swissinfo.ch iPad app is compatible with iPhone OS 3.2 and iOS 4.2, and it offers access to text, video and podcasts published in the swissinfo.ch website. Quick showcase of some application features:

- Access content in 9 languages: English, German, French, Italian, Spanish, Portuguese, Arabic, Chinese and Japanese.
- Store every article read and read it while offline.
- Get an overview of the news in the latest week.
- Add bookmarks for articles in a convenient location for later reference.
- Watch videos (this requires a network connection).

[1] http://itunes.apple.com/ch/app/swissinfo-ch/id407776916?mt=8
[2] http://www.swissinfo.ch/

- Listen to podcasts in the background while using other applications (requires a network connection and iOS 4.2).
- Print articles (requires iOS 4.2).
- Share articles con Twitter and Facebook.
- Share articles via e-mail, optionally with a PDF of the article.
- Change languages from within the application (might require a network connection to download content in the new language).

The swissinfo.ch team has also prepared a great, funny and viral video[3] to introduce the application! This video has been showcased in many websites and blogs all over the world and talks about the history of Switzerland from a rather unusual point of view :) Check it out!

You can download swissinfo.ch from the App Store![4] It is a free application.

[3] http://www.youtube.com/watch?v=nUH0r6VHUS4

[4] http://itunes.apple.com/ch/app/swissinfo-ch/id407776916?mt=8

Parmigiani Fleurier iPhone Application

Adrian Kosmaczewski

2010-12-28

akosma software is proud to announce the new and exclusive Parmigiani Fleurier iPhone application[1], developed in collaboration with RGB Productions[2]. This iPhone app is compatible with iPhone OS 3.2 and iOS 4.2 and offers the following features:

- Explore the exclusive catalog of Parmigiani Fleurier watches, including technical data and pictures.
- Read the latest news of Parmigiani.
- Find your nearest retailer using the integrated GPS.
- And explore a selection of exclusive bars, restaurants and hotels around the world, matching the exclusive Parmigiani lifestyle.

Parmigiani Fleurier is a free application and you can download it from the App Store[3]!

[1] http://itunes.apple.com/ch/app/parmigiani-fleurier/id406655736?mt=8
[2] http://www.rgbprod.com/
[3] http://itunes.apple.com/ch/app/parmigiani-fleurier/id406655736?mt=8

I Hate You, Airline Industry

Adrian Kosmaczewski

2010-12-28

I hate flying. I hate airplanes. I hate airlines. I hate crews. I hate ground handling teams. I hate everything that has to do with that shit. Deeply. Disturbingly. Profoundly.

I hate the way you airlines cram hundreds of people into the smallest of spaces. Do you really think my femur fits the distance between your seats? Do you really think I enjoy being pushed sideways for hours by my seat neighbor because the armrest is too narrow for the both of us? Do you really think I can eat my meal when the seat in front of me is in the horizontal position? Do you really think I can't avoid numb legs and feet during long flights? Do you really think I can go to the toilets without waking up all the people in the row in front of me or my neighbors?

I hate how long boarding and getting out of the damn plane takes. Haven't you noticed that airplanes usually have more than one door? Then why the fuck are all 380 passengers of a 747 getting into the plane though the same, unique, small door? Can't you design airports that take that into account? Can't you, jetty makers, airport designers, add an extension to boarding gates that goes above the wing or below the tarmac so that we can all get in and out through several doors at once?

I hate how you dare selling double tickets to obese people. Wouldn't it be better to have a couple of special seats in the front of the aircraft for them? You don't have any trouble overselling tickets and leaving people in the ground wondering what happened and begging you for a hotel voucher, but of course you can't plan in advance for the 5% of potentially obese passengers that have to endure your fucking shit. And let's not talk about families with kids, ok?

I hate your in-flight entertainment system. When it works (which, as per Murphy's Law, most often don't), your music sucks, your film choice is crappy, the sound is bad, and even worse, I hate how the captain interrupts my movie every 10 minutes to tell me that the outside temperature is about 3 degrees Kelvin or other nonsense that nobody fucking cares about, babbled through speakers that sound like if they were built in the 20s. To begin with, haven't you heard about that Dolby thing? And most importantly, don't you think we are already annoyed enough, to just shut the fuck up and fly this thing in time?

That's the only thing we care about, you moron: to get outta here as fast as possible.

I hate your crappy food. I hate how it tastes, I hate the bad manners of the crew members serving it, I hate that I can never have meat instead of pasta because I always happen to sit behind the person who got the last one and that yeah, you're very sorry about that. I'll have a Coke, please.

I hate how airline websites fail big time. I hate how I have to always spend longer than required to find what I'm looking for, that your search engines are useless, that I have to spell correctly the codes of the airports, that your date picker is unusable without Flash or JavaScript, that the back button resets the whole form, that your animated intro annoys me every time I want to fucking spend money on your idiotic company because you happen to be the only idiots flying where I need to go. And airport websites are not better, so here goes a message to those dear airport webmasters: I want to know, right now, fast, without any more required clicks, if my flight is delayed, canceled or in time. I do not, let me repeat, I DO NOT CARE about how nice your first class lounge is; I will most probably never use it. The same goes for any other information. Put it behind a menu and don't bother me. Thank you.

I hate how I get the same crappy level of service when I pay 25 bucks for a 2 hour trip to Madrid or when I pay 2000 dollars for a roundtrip flight to Argentina that lasts 14 hours. Are you fucking kidding me? What is your problem, you dickhead? Do you really think I do not see how you are fucking filling your pockets with my cash?

I hate how inaccessible, unfriendly, broken and even expensive, airports are. I hate how immigration booths are all closed but one, and you spend more time waiting to show your passport than in the flight. I hate how your tax-free shit shops are more fucking expensive than downtown shops, and how they shamelessly pretend to have the best prices on Earth. Do you really think I was born yesterday?

I hate the mind-boggling algorithms I have to execute in order to know which terminal my flight is leaving from. It goes something like this, starting with the basic questions, domestic or international? Air Exhaust or Air Compression? Oh, then it's terminal G, door 257. You must enter through terminal N and then take our new air-magnetic-levitation-superconductor-enabled-robot-train and get out at terminal H, then walk through the panoramic gateway above the tarmac, and then you'll see the checkin booths at your left. Oh, since it's a code-sharing flight, you must use the booths of Blowjob Airlines to check in, then pay the airport tax in counter 734 and proceed through security and later through passsport control to gate Y35, but hurry up, your flight is boarding right now.

I hate how airport terminals are miles away from each other and how bad they are referenced and how hard it is to understand your information panels. Haven't you noticed that small airports are usually faster

to get in and out, have shorter distances between the plane and the terminal, people board using both doors and even better, are easier to get to from cities? The solution is not having two- or three-stories tall planes carrying 800 people at once[1]; STOP THAT SHIT. That won't work. If your airports can't handle 200 people at once per plane, do you really think you can handle more? Really?

I hate how connecting flights are always clutching at straws. A small delay in a flight, a longer queue in the security checks or even the fact of having to recheck-in on the new flight (because some airlines can't access the computers of each other in order to check you in all legs at once before departure), and your flight is gone. And if you are really unlucky, you will see the door of the gate being closed in front of you as you sweat your way to it, together with the grins of the ground team looking at you. You are then left to pray that you won't have to pay for a new ticket, that you will get a hotel for that night, and that all the shit printed in the "passenger rights" posters behind the counter is true. By the way, showing those posters implicitly tells me that something has gone really wrong with your industry.

I hate your security controls that don't protect anyone, that don't prevent anything, that just annoy and harass everyone. I hate your assaults on my personal sphere. I mean no harm to you. Leave me the fuck alone with your security pricks. I hate listening to the same security information every fucking time we take off, about how to put my oxygen mask or how to fasten my seat belt. The airline industry might have a lower number of accidents than other forms of transportation, but when you are involved in a plane crash, the odds of getting alive are lower than in the highway. No wonder sometimes people applaud when planes land; we just don't trust you to get us there alive.

I hate the inhuman conditions you airlines make your crews work in. I hate how they have to strike in order to have some attention, while you fucking MBAs running these companies get big bonuses at the end of the year. Because when you treat your employees like shit, they spit on my coffee, you shithead. They work overtime, they try to do a living in the worst of industries, and you treat them like shit. No wonder they get in strike.

I hate how you fucking dare losing my bags. I hate how I have to cross my fingers every time I travel to avoid having them sent to Timbuktu or Novosibirsk. Don't you see the tags with the airport codes and the barcodes printed in them? And, even after losing them, is it really that difficult to send it faster than 3 days later to their owner? Really? Do you really think I will buy new clothes every time I travel just because baggage is handled by pathetic monkey-like systems unable to read correctly a tag? Oh, but of course, you will tax me for every extra kilo in those same bags like if I was carrying gold bars. Fuck you.

I hate how everything is a good reason to be late, or to not fly at all. Snow. Strikes. Rain. Late connections. UFOs. Other planes.

[1] http://en.wikipedia.org/wiki/Airbus_A380

Storms. Winds. Birds. Clouds. Thick air. Thin air. Engines. Flaps. Eyjafjallajökull[2]. Wings. Terrism[3]. Airport facilities. Tires. Oil. Gravity. Mountains. Plains. Seas. Passengers. Bags.

I hate how a plane can disappear in the middle of the ocean without a trace[4]. Haven't you heard about this thing called a satellite? Can't you have a direct, permanent link with a satellite, so that in case of accident you can be notified milliseconds, not hours, later? We are in 2011, you fucking murderers. Black boxes were a neat idea in 1924, shouldn't you be upgrading that thing anytime soon?

In other words: WHAT IS YOUR PROBLEM? If you are unable to provide a service, well THEN DON'T DO IT. Do I provide healthcare? Am I a lawyer? Do I own a grocery store? No, because I know shit about those professions. BUT I DON'T PRETEND TO EITHER.

I do not trick people with nice advertising showing how big your first class seats are (probably the most useless kind of advertising ever). I do not fill my mouth with useless shit about your commitment to service. I do not lie to people about what I do and how I do it. Be frank: say that your service is as bad as anyone else. Say it. Admit it. Be as much as a failure as you want, but please, don't be hypocrite.

The airline industry is deeply broken. It must be redesigned from scratch. If you are reading this and you happen to be the CEO of one of those fucking airlines, then please know that I wholeheartedly hate you, that you and your company are worthless, and that you have won the Guiness record for making the most millions of unhappy people per minute. Go to hell.

[2] /blog/making-traveling-enjoyable-again/

[3] http://www.urbandictionary.com/define.php?term=terrism

[4] http://en.wikipedia.org/wiki/Air_France_Flight_447

Retrospective 2010

Adrian Kosmaczewski

2010-12-29

2010 is coming to an end, and it was an excellent year for akosma software! We would like to thank all of our clients, service providers, partners and friends for the trust and the business! We wish you a great 2011 and we remain at your service as your partner of choice for your software needs.

We are thrilled and proud to share with you a few highlights of 2010 below:

iOS Projects

To put it bluntly, during 2010, the business around iOS has simply exploded. The iPad has brought a tremendous amount of interest, money and business to the iOS platform, where many companies are looking for ways to provide services, leisure and information through the hundreds of millions of iPhone, iPad and iPod touches available throughout the world. Games, newspapers, brands, service providers: having an iPhone app is as important as having a website!

These are the applications developed directly or jointly by akosma software, released through the App Store or the web for worldwide distribution:

[1]

[1] http://itunes.apple.com/dk/app/danske-mobilbank/id388423462?mt=8

iPhone

This iconic and magnetic device amazes and fascinates more and more people every day, and sets the pace of the whole industry:

- Danske Bank[2], the most important banking app in Scandinavia, which has been #1 best seller for months and one of the most popular iPhone applications of 2010. This incredible application was developed jointly with Trifork GmbH[3].
- Parmigiani Fleurier[4], the Swiss watchmaker, has published a gorgeous iPhone application showing their exclusive collection of watches. This app has been jointly produced by akosma software and RGB Productions[5].
- Mnesia[6], a new concept in to-do lists, introduced by Trifork Switzerland GmbH[7].
- Thierry Weber[8], one of the most prominent european podcasters, published his own iPhone application too!
- Senbei[9], the one and only iPhone client for the groundbreaking Fat Free CRM[10] open source application by Michael Dvorkin[11].

[12]

iPad

The iPad has been a tremendous success, and not only has akosma software brought to Switzerland the first iPads[13] in April, we have also

[2] http://itunes.apple.com/dk/app/danske-mobilbank/id388423462?mt=8
[3] http://trifork.ch/
[4] http://itunes.apple.com/ch/app/parmigiani-fleurier/id406655736?mt=8
[5] http://www.rgbprod.com/
[6] http://itunes.apple.com/ch/app/mnesia/id392269920?mt=8
[7] http://trifork.ch/
[8] http://itunes.apple.com/ch/app/thierryweber-com/id342010688?mt=8
[9] http://itunes.apple.com/ch/app/senbei/id357417115?mt=8
[10] http://fatfreecrm.com/
[11] http://twitter.com/mid
[12] http://itunes.apple.com/ch/app/swissinfo-ch/id407776916?mt=8
[13] /blog/first-ipads-in-switzerland/

been actively involved in important projects for this device:

- swissinfo.ch[14], the iPad application of the official Swiss news broadcasting service, introduced by a great viral video[15] that has been showcased in many different news services, like TUAW[16], CNET[17], 20 minutes[18] and the Huffington Post[19]!
- digital2.0[20], a collaboration between Moser Design[21], VPS Prod[22] and akosma software, serves as an introduction to the many capabilities of this new device.

[23]

Universal (iPhone & iPad)

The creation of “Universal Apps” for both the iPhone and the iPad has special quirks, particularly when targeting iPhone OS versions prior to 4.x; we have published a popular article explaining our findings and our techniques[24] to streamline the creation of such applications.

- Le Temps Sortir Dîner[25], an exclusive restaurant guide created for the Geneva-based newspaper Le Temps, developed jointly with Trifork GmbH[26].

[14] http://itunes.apple.com/ch/app/swissinfo-ch/id407776916?mt=8
[15] http://www.youtube.com/watch?v=nUH0r6VHUS4
[16] http://www.tuaw.com/2010/12/17/found-footage-ipad-app-delivers-humorous-look-at-swiss-history/
[17] http://news.cnet.com/8301-17852_3-20026290-71.html
[18] http://www.20min.ch/digital/dossier/apple/story/Wilhelm-Tell-heiratet-Heidi-20958738
[19] http://www.huffingtonpost.com/2010/12/15/animated-history-switzerland_n_797133.html
[20] http://itunes.apple.com/ch/app/digital2-0/id379896829?mt=8
[21] http://www.moserdesign.ch/
[22] http://vpsprod.com/
[23] http://www.letemps.ch/tout_le_temps/applications/sortir_diner/
[24] /blog/migrating-iphone-3-x-apps-to-ipad-and-ios-4-0/
[25] http://www.letemps.ch/tout_le_temps/applications/sortir_diner/
[26] http://trifork.ch/

Web App Optimized for iPhone & iPad

Web apps, based on HTML5, JavaScript and CSS3, can be customized and enhanced for iOS devices; we have teamed up with our partners and friends of Zerofee[27] in London to create a unique web app for the iOS platform:

- Dezeen Watchstore Web App[28], featuring an innovative and timeless watch design.

Consulting and Training

In akosma software we provide many services, among which consulting takes a place of choice; we've provided countless hours of guidance to many businesses in Switzerland and abroad, explaining them various aspects of the distinct dynamics of the iOS platform. Among those clients (many of which have asked us to remain anonymous), we have worked closely with:

- Bean Creative[29], from Virginia (USA) for the game Super Why![30]
- Rudicubes[31], from Geneva (Switzerland) for their game Cubz[32].
- iOS development training for professionals, developers and the general public; we have been steadily providing training services to hundreds of people during the year!

[33]

Conferences

During 2010, akosma software has both organized and participated as speaker in many conferences and technical events accross Europe:

- Scandinavian Developer Conference 2010[34] (Göteborg, Sweden)
- Intro to iPad for iPhone Developers[35] (Zürich, Switzerland)
- Dev Days for iPhone[36] (Geneva, Switzerland and London, UK)
- JAOO 2010[37] (Århus, Denmark)

[27] http://zerofee.org/
[28] http://dezeenwatchstore.com/clock/
[29] http://www.beancreative.com/
[30] http://itunes.apple.com/us/app/super-why/id357422351?mt=8
[31] http://playcubz.com/
[32] http://itunes.apple.com/app/cubz-3d-sliding-puzzle/id365754207?mt=8
[33] http://gotocon.com/
[34] http://www.scandevconf.se/2010/
[35] /blog/ipad-for-developers-presentation/
[36] http://devdayforiphone.com/
[37] /blog/jaoo-2010/

- Rezonance First[38] (Yverdon Les Bains, Switzerland)
- Talk at EPFL[39] (Lausanne, Switzerland)

[40]

Open Source

akosma software is actively involved in the open source arena, giving back to the community as much as we can, because we strongly believe that we all benefit from sharing code and contributing to each others' projects. Here are some of the projects that we have published or updated during 2010:

- Senbei[41]
- cortito[42] (installed in akos.ma[43] + myze.ro[44])
- nib2objc[45]
- iPhone Web Services[46]
- Core Text Objective-C Wrapper[47]

[48]

[38] /blog/applications-iphone-il-est-temps-de-s\XeTeXglyph\numexpr\XeTeXcharglyph"0027\relax{}y-mettre/
[39] /blog/talking-at-the-epfl-today/
[40] http://itunes.apple.com/ch/app/senbei/id357417115?mt=8
[41] http://projects.akosma.com/projects/senbei
[42] http://projects.akosma.com/projects/cortito
[43] http://akos.ma/
[44] http://myze.ro/
[45] http://projects.akosma.com/projects/nib2objc
[46] http://projects.akosma.com/projects/iphone-web-services
[47] /blog/core-text-objective-c-wrapper/
[48] http://www.culturepod.ch/

Interviews

As active members of the iOS development community in Switzerland, we've been interviewed in several occasions during the year, to provide our opinion about the status of the iOS software landscape in this country and abroad:

- CulturePod.ch interview[49] (Thanks to Thierry Weber[50]!)
- First iPads in Switzerland[51] (Thanks again to Thierry Weber[52]!)
- iPhoneDevSuisse interview[53] (Thanks Junior[54]!)
- Le Potecast #21[55] (Thanks yet again Thierry Weber, Jay-Rôme Berthoud[56] et Nicolas Pittet[57]!)
- Article about the Geneva Dev Days for iPhone sur Le Temps[58] (Thanks to Anouch Seydtaghia!)
- Article on Com.in magazine[59] about Mobile App Stores (Gracias a Victoria Marchand[60]!)

[61]

Other News

But that's not all! Many more good things have happened in akosma software during 2010:

[49] /blog/culturepod-ch-interview-of-adrian-kosmaczewski-by-thierry-weber/
[50] http://www.thierryweber.com/fr/
[51] /blog/first-ipads-in-switzerland/
[52] http://www.thierryweber.com/fr/
[53] /blog/interview-by-iphonedevsuisse/
[54] http://www.bonto.ch/
[55] /blog/le-potecast-21/
[56] http://www.chicheux.ch/
[57] http://www.inthebox.ch/
[58] /blog/dev-day-for-iphone-sur-le-temps/
[59] /blog/article-dans-com-in-magazine/
[60] http://www.cominmag.ch/mag/author/victoria-marchand/
[61] http://muchasnotitas.com/

- Marco Arment[62], the creator of Instapaper[63], has added support for exporting[64] items to Notitas[65]!
- Some of our code has been included in Kevin Smith's iPhone app[66]!
- We've been notified that cortito[67] is being used internally by ATT Interactive[68]!
- We have been asked by Pragmatic Programmers[69] to provide technical reviews and also a praise for the recently released "iPad Programming"[70] book, by Daniel Steinberg and Eric Freeman!
- Last but definitely not least, nib2objc[71] has been featured by Erica Sadun in her "iPhone Developers Cookbook"[72]!

We wish you the best for 2011!

[62] http://www.marco.org/
[63] http://www.instapaper.com/
[64] http://blog.instapaper.com/post/1654586873
[65] http://muchasnotitas.com/
[66] http://www.denvog.com/iphone/KevinSmith/index.html
[67] https://github.com/akosma/cortito
[68] http://attinteractive.com/
[69] http://pragprog.com/
[70] http://www.amazon.com/iPad-Programming-Daniel-H-Steinberg/dp/1934356573
[71] http://projects.akosma.com/projects/nib2objc
[72] http://www.amazon.com/iPhone-Developers-Cookbook-Building-Applications/dp/0321555457

Conferences 2011: OOP, QCon and SDC

Adrian Kosmaczewski

2011-01-04

2011 has just started and we already have a couple of exciting news: we are humbled and thrilled to announce our participation in some awesome conferences throughout Europe!

[1]

- In January I will be speaking in the OOP 2011 Conference in Munich[2], which spans from Monday 24th to Friday 28th! This is the 20th edition of one of the most important software conferences in Europe, featuring this year speakers like Martin Fowler, Erich Gamma, Tom DeMarco, Scott Berkun and Kevin Henney! My session is called "Introduction to iOS Software Development"[3] and will take place on Thursday, January 27th, at 9am.
- In March, it will be the turn of QCon London 2011[4], where I will be hosting the mobile track[5], with the confirmed participation of Graham Lee, talking about mobile app privacy[6]! That will happen on Wednesday, March 9th.

[1] http://www.oop2011.de/

[2] http://www.oop2011.de/

[3] http://www.sigs-datacom.de/oop2011/konferenz/sessiondetails.html?tx_mwconferences_pi1%5BshowUid%5D=382&tx_mwconferences_pi1%5Bpointer%5D=0&tx_mwconferences_pi1%5Bmode%5D=1&tx_mwconferences_pi1%5Bs%5D=0

[4] http://qconlondon.com/london-2011/

[5] http://qconlondon.com/london-2011/tracks/show_track.jsp?trackOID=417

[6] http://qconlondon.com/london-2011/speaker/Graham+Lee

- Finally, in April I will be at the Scandinavian Developer Conference 2011[7] in Göteborg, in Sweden, talking about the mobile web[8]. This conference will take place between Monday 4th and Tuesday 5th.

I look forward to meeting you there in person! Don't hesitate to stop me and say hi if you attend any of these absolutely fantastic events.

[7] http://www.scandevconf.se/2011/
[8] http://www.scandevconf.se/2011/conference/speakers/arian-kosmaczewski/

Advanced iOS 4.2 Training Course – Zürich, February 7th and 8th 2011 – Enroll now!

Adrian Kosmaczewski

2011-01-21

akosma software is happy to announce the Advanced iOS 4.2 Training Course[1] jointly organized with Trifork GmbH[2]. This course requires that the attendees are familiar with Objective-C and have developed at least a few simple iPhone applications. The attendees should bring their own MacBook, MacBook Air or MacBook Pro with the latest iOS SDK installed. This class will take the attendees through the following subjects:

[3]

Day 1: Universal apps

- Building iOS 4.2 universal applications compatible with version 3.x of iPhone OS
- Advanced user interface design for the iPhone and the iPad
- Integrating your application with Twitter, Facebook and other social networking sites
- Performance enhancements using Core Foundation in your applications

Day 2: Performance

[1] http://events.linkedin.com/Advanced-iOS-4-2-Training-Course/pub/519325
[2] http://trifork.ch/2011/02/07/ios-4-2-training-course/
[3] http://events.linkedin.com/Advanced-iOS-4-2-Training-Course/pub/519325

- Using Instruments to find performance problems
- Major ninja-level optimizations
- 2D graphics and animation: Quartz and Core Animation
- Video, audio, the iPod library, AirPlay, etc.

About this training:

- Language: English
- Place: Technopark Zürich[4]
- Date: 7-8 February 2011
- Price: CHF 1400
- Max seats: 20
- Registration: Please send name, company and billing address to Serife Cakmak[5].

[4] http://www.technopark.ch/start.cfm
[5] mailto:sec@trifork.com

14 y 20 años después

Adrian Kosmaczewski

2011-01-24

El 3 de diciembre de 1996 volví a Argentina después de una ausencia de casi 6 años. Volví para buscar un padre, unos amigos, una realidad que estaba plasmada en un pasaporte argentino, del que me costaba entender el sentido.

14 años después de aquel momento, hice un viaje de geografías similares, pero esta vez las circunstancias fueron tan distintas. Estoy casado y fui con mi mujer; profesionalmente encontré una veta interesante, y creo que como persona crecí, al menos, lo necesario como para sobrevivir en esta tierra desolada. No encontré todas las respuestas, pero se que eso no se logra en el espacio de una vida. Hay cosas que requieren mas tiempo.

Pero por sobre todo, la gran razón de este viaje fue mamá. Su muerte, hace ya casi un año, nos dejó a todos un poco en la pampa y en la vía. La idea de llevar sus restos a Argentina nos surgió con Claudia mientras tramitábamos su cremación. En pleno estado de shock. Pero creo que fue una buena idea, ya que un par de meses antes la vieja me decía que quería irse a pasar sus últimos años allá. Creo que ella ya sabia que el fin estaba cerca.

Y en algún punto, me parece que yo también sentía que le quedaba poco tiempo de vida. Su debilitamiento fue progresivo pero imparable, y duro un año entero.

El hospital ya no era una opción valida. Las ultimas veces que estuvo ahí fue cada vez peor. Cuando después de su muerte, fuimos con Clau a ocuparnos de su departamento, encontramos una cantidad de medicamentos espantosa, muchos de los cuales servían solamente para eliminar los efectos secundarios de los otros. No era una vida. Era una esclavitud química, una muestra más de que, como sociedad, simplemente estamos muy equivocados en nuestra manera de enfrentar la enfermedad y la muerte. Muy equivocados.

Ahora los restos de mamá son uno solo con el océano Atlántico. Aquel mismo océano cuyas aguas bañaban su tobillo dañado, aquel que la hacía suspirar, aquel donde pasó sus últimas verdaderas vacaciones en el ’85. Ella y el océano son uno para siempre, con color, sin dolor[1].

[1] /blog/color-sin-dolor/

Volver a Argentina, entonces, fue cerrar ese capitulo que se abrió el 20 de febrero de 1991. Hace casi 20 años, día por día. En algún punto parece que fue ayer, por otro lado aquel viaje suena tan remoto y lejano. Pero pasó, fuimos nosotros, o al menos esas fotos están ahí. Difícil saberlo.

Esa fue nuestra vida, nuestra elección. Buena o mala, fue en todo caso lo que le dio forma a mi vida. Mamá es responsable de todo? No, yo tomé gran cantidad de decisiones propias, que no tienen nada que ver con ella. Y bastante disgustos y angustias le causé. Creo que ella se dejo ir en un momento en el que vio que había logrado cierto equilibrio, cierta tranquilidad.

Estar en Suiza, paradójicamente, fue lo que me permitió encontrarme con Claudia. Los viajes cambian vidas, transforman las existencias, de maneras insospechadas, de formas delirantes y maravillosas.

Gracias, vieja, porque a pesar de todo los quilombos, de no haber sido tan vos misma, mi vida no sería lo que es. Y estoy contento de ser quien soy. Te quiero mucho.

Wordle

Adrian Kosmaczewski

2011-02-08

Word cloud created with Wordle[1].

[1] http://www.wordle.net/

nib2objc Featured on The Changelog

Adrian Kosmaczewski

2011-02-17

nib2objc[1] has been featured recently in The Changelog[2], a blog about open source projects:

> Interface Builder is one of the coolest things about Cocoa development. Being able to draw your interfaces visually can save you tons of otherwise tedious code to create layouts and set visual styles for your user interface elements.
>
> Adrian Kosmaczewski, iPhone and iPad developer and creator of Device DNA and several other apps in the App Store turned us on to nib2objc, which is a brilliant name because it takes NIB (or XIB) files from Interface Builder and converts them to Objective-C.

[1] http://github.com/akosma/nib2objc

[2] http://thechangelog.com/post/3077537300/nib2objc-set-of-tools-and-utilities-command-line-gui-mac

20 Years Ago

Adrian Kosmaczewski

2011-02-20

Exactly 20 years ago, on Wednesday February 20th, 1991, my mother and I arrived to Geneva, Switzerland, from Buenos Aires, Argentina.

It was a rainy, gloomy day, just like today. Our KLM flight from Amsterdam landed in Geneva airport at around 5pm. We left our bags in the lockers at the airport's train station, and then took the first train to Geneva. I knew, from reading tourist guides, that there was a tourist office in Geneva's main train station, so we decided to go there first, get a hotel, and then bring the bags later from the Airport.

You might wonder why we were looking for a hotel, when actually the idea was to come and live in this city; good question. The thing is that we knew nobody in here, and even better, nobody knew we were here. When I say nobody, I mean not even my mother's brother, or a couple of cousins, both in the French- and German-speaking parts of Switzerland. Our phone calls from Argentina to Switzerland to tell someone about us coming were a failure, mainly because, well, the phone numbers we had were no longer valid. My mother had not been here since 1964, and even worse, she had lost most contact with the family here after my grandmother died in 1985. In my case I was here for the first time.

So, given that we were all by ourselves, our priority was to find a place in this city, our new home. First a hotel, then to get a job and rent an apartment. We finally booked a room in the "Hotel Adris", a rather awful but cheap hotel in Rue Gevray (it does not exist anymore), in the Pâquis neighborhood in downtown Geneva, 7 blocks away from the station, and just 150 meters away from the lake.

Little did we know that we would live in and around Pâquis for the next 3 years and a half.

When I look back, I really think we were dead crazy. The only thing that guaranteed us not being sent home by the police was our Swiss passports, because, frankly, at that time I didn't even speak French fluently - and my mother had forgotten hers in almost 30 years of not speaking it every day. We were quite lost, as a matter of fact.

I remember our first evening in Switzerland. We were very tired after almost 24 hours of flying (EZE -> AMS -> GVA), but even so, after

leaving the bags in the room we decided to tour around, just as the night was coming up.

We finally ended up having a pizza in a nearby restaurant. It was “La Grappe d’Or”, and it is still there. We had a pizza, we talked about what we had seen so far, and we planned the next steps.

Needless to say, my life changed completely that day.

Things went quite fast the following days. We spent Thursday and Friday going to real-estate offices looking for an apartment, but the thing was that they wouldn’t rent us one because my mother did not have job, and on the other hand, she couldn’t get a job without an address.

Our first weekend in Geneva was a rather stressful one. We were effectively stuck in a quite awful situation, without any solution in sight. Most of the so called “family members” of my mother didn’t even bother returning our phone calls, or were just telling us to go back to Argentina. What you just read. The only person that was really worried about us, and ended up coming to Geneva in March to meet us, was her cousin Hella from Schaffhausen, and until my mother’s death, she and her family have been the only ones with whom we have kept contact. They are our Swiss family, without any doubt.

Finally, on Monday morning my mother broke down in tears in front of a woman in the offices of Pilet & Renaud, a real estate agency in the Place du Cirque. We didn’t have much money, and we could not stay much longer in a hotel, so finding an appartment was top priority for us. This woman, whose name I don’t remember, was touched by our story, and told us to go to the “Hospice Général”, a help institution in Geneva, which has an office for Swiss people returning from abroad.

I never understood why the Swiss consulate in Buenos Aires never told us about that office in the first place. Anyway.

On Monday afternoon we got help from the Hospice Général, in the form of a warrant. With it, we rented our first apartment on Tuesday, and we moved in on Wednesday, exactly one week after we arrived. We didn’t have furniture, and our only clothes were in those bags, but the first step was taken. We were living in Geneva. Our first address was Rue Charles-Cusin 10, 1201 Genève. We rented a small studio on the first floor, right above the kebab shop in the corner of Rue de Monthoux.

I remember that we celebrated this by having dinner in the “Auberge de Savièse”, not far from the apartment. The following steps were a job, and also a school; I had to finish my secondary school.

My mother and I managed, against all odds, and without any other help, to build a life from scratch[1] and fulfill all of those steps, and many others, in only a couple of months.

[1] /blog/blessed/

Exactly 20 years ago.

New Address

Adrian Kosmaczewski

2011-02-24

Update your address books! akosma software is moving to a new address:

Chemin du Chaney 9
1610 Oron-la-Ville
Switzerland

Our telephone, web, e-mail, Skype and other contact information[1] stays the same.

Funny fact: if you search this address in Google Maps you won't find it, because the street is misspelled, and it appears as "Chanay" instead of "Chaney" (in French, both are pronounced similarly, which might explain the mistake). I've already notified the Google Maps team about this problem, and they acknowledged the issue, so they will update the tiles soon. In the meantime, here's the real location of our new office, thanks to the Single Google Map plugin for WordPress[2]:

[1] /about/
[2] http://clarknikdelpowell.com/wordpress/simple-google-map/

Smart Pointers in Objective-C++

Adrian Kosmaczewski

2011-03-28

One of the coolest features of C++ are templates, of which I've been drooling in the past[1]. One of the most useful things that templates have brought to C++ are smart pointers, which simplify memory management tremendously; they combine the capacity of C++ to instantiate objects in the stack, the flexibility of heap allocation, and template classes, all in one thing.

I've talked about them in a previous article in this blog[2]. In C++, a smart pointer will automatically call "delete" on the managed pointer when it goes out of scope, simplifying resource management and providing many more advantages over common pointers, as Andrei Alexandrescu explained in chapter 7 of his book "Modern C++ Design"[3].

In Objective-C, the "new" and "delete" keywords are replaced by some combination of "alloc / init", "copy", "release", and "autorelease". But given that Objective-C does not allow for stack allocation of objects (apart from blocks, but that's another story[4]), I've tried to create such a beast in Objective-C++.

These are the features of my smart pointer:

- It takes ownership of any non-autoreleased object with a retain count of at least 1.
- It has value semantics, that is, it is copied by value from stack frame to stack frame, while the "owned" object, as any Objective-C entity, lives happily in the heap.
- When it goes out of scope, its destructor is called, which sends a "release" message to the owned object.

Here's how you could use it:

```
for (NSInteger index = 0; index < 10; ++index)
{
    // A SmartPointer around an NSObject
```

[1] /blog/templates/

[2] /blog/how-knowing-c-and-c-can-help-you-write-better-iphone-apps-part-1/

[3] http://www.amazon.com/Modern-Design-Generic-Programming-Patterns/dp/0201704315

[4] http://www.mikeash.com/pyblog/friday-qa-2010-01-15-stack-and-heap-objects-in-objective-c.html

```
    SmartPointer<NSObject> obj = [[NSObject alloc] init];

    // SomeType is a typedef (see above)
    SomeType someObj = SomeType::create();

    // You can get access to the underlying Objective-C object
    // using "*", ".get()" or "()"; they all return the same pointer.
    // And once you have it, it's a normal Objective-C pointer
    // you can send messages to:
    SmartPointer<NSMutableArray> array = [*someObj createArrayWithCapacity:5];

    // Here a SmartPointer around an NSNumber
    NSNumber *value = [[NSNumber alloc] initWithInt:1424];
    SmartPointer<NSNumber> number = SmartPointer<NSNumber>(value);

    // Playing with the array we got above, just to show it's a normal object
    [array.get() addObject:@"test1"];
    [*array      addObject:@"test2"];
    [*array      addObject:number()];
    [array()     addObject:*obj];

    // After this NSLog call, in the console you should see
    // "[SomeClass dealloc]" which is printed when the SmartPointer goes
    // out of scope, sending the release message on the underlying pointer.
    NSLog(@"array %@", *array);
}
```

In general, NSAutoreleasePools fit the bill quite comfortably in Objective-C for this kind of tasks, apart from some performance issues in iOS devices (particularly older iPhones and iPod touch devices), but I think there might be cases where just using a "non-autoreleased" approach to resource management might be useful, and typing an extra "[obj release]" line would be too much to ask :)

Of course, this is just an experiment, that I just publish as part of my own curiosity. I would be more than glad to hear some feedback about it from people more knowledgeable in Objective-C and C++ than I am. The code, as usual, is available in Github[5] under a liberal BSD license. Enjoy!

Update, 2011-03-28: just found by pure chance another, much more complete[6], implementation of this idea (note to self: always Google first!) with its code in Github[7].

[5] https://github.com/akosma/SmartPointerTest

[6] http://www.levelofindirection.com/journal/2010/8/13/ocptr-a-smart-pointer-for-objective-c.html

[7] https://github.com/philsquared/OCPtr

Talking at the Scandinavian Developer Conference 2011

Adrian Kosmaczewski

2011-03-31

Next week I'll be speaking at the Scandinavian Developer Conference 2011[1]! This year I will be giving the following two talks[2] on Monday April 4th, in the Mobile track:

- **Mobile Web Rising**: The popularity of smartphone devices and platforms like the iPhone, Android and BlackBerry has triggered an explosion of proprietary and mutually incompatible software platforms. Web applications, thanks to HTML5 and the WebKit engine, are slowly gaining traction as a true cross-platform environment, suitable to reduce development, deployment and quality management costs. The aim of this talk is to provide an overview of the current state of mobile web development, including an exhaustive survey of the most important standard technologies available today to create compelling, immersive user experiences on mobile devices. The talk is suited to desktop, web and mobile developers looking to expand the reach of their applications.
- **Integrating iPhone Applications with Backend REST Services**: In a world of interconnected services, any iPhone application must be able to access online services to provide a rich user experience, raising concerns of compatibility, security, and performance. This talk will provice an overview of the techniques required to connect a native iPhone application, developed using Apple's iPhone SDK, to backend REST web services, highlighting different approaches and tradeoffs.

If you are going don't hesitate to stop me and say hi! I would love to meet you there.

[1] http://scandevconf.se/

[2] http://scandevconf.se/2011/conference/speakers/arian-kosmaczewski/

Best Books of 2010

Adrian Kosmaczewski

2011-03-31

It is that time of the year again, just like in[1] previous[2] years[3]. This is the list of the books I enjoyed most in 2010! You know that I like reading at least 6 books per year, and learning a new programming language every year. Last year's programming language was LISP[4], and the books, well, here they go.

eBooks

By all means, it is clear that 2010 was the year of the eBook. Maybe it's because of the iPad, but I've been consuming more and more eBooks, even if I still enjoy buying some classics in paper form. Kindle, iPad, iBooks, Nook, GoodReader, PDF, ePub, all of those names have shaped my way of reading last year.

But one of the most visible changes of switching to eBooks was the speed of reading; consuming eBooks is fast, much faster than reading normal books. I can't say that I prefer one or the other; it's simply different. But reading eBooks is faster than reading paper books. Probably there's a warmth factor in paper books, which makes me enjoy them longer, I don't know, but the fact is, in 2010 my book reading consumption has gone up in an alarming rate.

Reading as a Reviewer

Another big change in 2010 was that, for the first time, I've been asked to review a book before it's published, as a technical reviewer; and boy, what a book: it was iPad Programming by Daniel Steinberg and Eric Freeman[5]. I've started reviewing it when the iPad's iPhone 3.2 SDK was still in beta, and I remember it was the first book I've read in iBooks on my iPad. It's an awesome reference for iPhone developers who want to start developing apps for this (at the time) new device, providing both design and programming techniques.

[1] /blog/best-books-of-2007/
[2] /blog/best-books-of-2008/
[3] /blog/best-books-of-2009/
[4] https://github.com/akosma/PracticalCommonLisp_ePub
[5] http://www.amazon.com/iPad-Programming-Daniel-H-Steinberg/dp/1934356573

Actually, I got the great chance to meet Daniel in person later during the DevDay for iPhone in London and Geneva[6] later, and he's not only a great author but now too a good friend of mine.

Physical Books

But not everything was eBooks in 2010; I've been buying paper books as well, which I love and enjoy a lot; my preferred, by all standards, is this gem called Cocoa Design Patterns[7] by Erik Buck and Donald Yacktman, which I think is a mandatory read for any iOS or Mac OS X developer; it is probably one of the most important books ever written about Objective-C.

Buck and Yacktman are also the authors of the 2002 Cocoa Programming[8] absolute reference I've talked about[9] in 2005, so I'm not surprised that this new collaboration yields such an impressive volume.

Other gems I've read this year:

- REWORK[10] by Jason Fried and David Heinemeier Hansson;
- Presentation Zen[11] by Garr Reynolds;
- HTML5 For Web Designers[12] by Jeremy Keith (A Book Apart).

More eBooks

Finally, the list of great eBooks I've read, in either ePub, PDF, Kindle or just plain websites, in no particular order:

- 17 Rules Successful Companies Use to Attract and Keep Top Talent[13] by David Russo;
- Don't Make Me Think![14] by Steve Krug;
- The iPhone Developer's Cookbook[15] by Erica Sadun, 2nd edition;
- iPhone and iPad Apps Marketing[16] by Jeffrey Hughes;
- Practical Common Lisp[17] by Peter Seibel, which I adapted to ePub[18] format;

[6] http://devdayforiphone.com/
[7] http://www.amazon.com/Cocoa-Design-Patterns-Erik-Buck/dp/0321535022
[8] http://www.cocoaprogramming.net/
[9] /blog/my-bookshelf-part-i/
[10] http://37signals.com/rework/
[11] http://www.amazon.com/Presentation-Zen-Simple-Design-Delivery/dp/0321525655
[12] http://www.abookapart.com/products/html5-for-web-designers
[13] http://www.amazon.com/Rules-Successful-Companies-Attract-Talent/dp/0137146701
[14] http://www.sensible.com/dmmt.html
[15] http://www.amazon.com/iPhone-Developers-Cookbook-Building-Applications/dp/0321659570
[16] http://www.amazon.com/iPhone-iPad-Apps-Marketing-Biz-Tech/dp/0789744279
[17] http://www.gigamonkeys.com/book/
[18] https://github.com/akosma/PracticalCommonLisp_ePub

- _why's poignant guide to Ruby[19] by _why the lucky stiff, also available as ePub[20] and PDF versions[21];
- 20 Things I Learned About Browsers and the Web[22] by the Google Chrome team.

[19] http://mislav.uniqpath.com/poignant-guide/
[20] https://github.com/sorah/poignant-guide-epub/
[21] http://www.rubyinside.com/media/poignant-guide.pdf
[22] http://www.20thingsilearned.com/home

Presentations at SDC 2011

Adrian Kosmaczewski

2011-04-30

These are the presentations I've given at the Scandinavian Developer Conference 2011 in Göteborg, Sweden: "Integrating iOS Applications with Backend REST Services" and "Mobile Web Rising".

Presentation at QCon London 2011

Adrian Kosmaczewski

2011-04-30

This is the presentation I've given in QCon London 2011: "Introduction to iOS Software Development"

Presentation at OOP 2011

Adrian Kosmaczewski

2011-04-30

This is the presentation I've given in the OOP Conference 2011 in München, Germany: "Introduction to iOS Software Development".

immedia: the leader mobile solutions provider in South Africa

Adrian Kosmaczewski

2011-05-31

akosma software was born with the dual objective of providing world-class software solutions and technical advice to companies, all over the world. One of the most unexpected and delightful outcomes of our work in akosma is the incredible friends we find along the way, every day.

[1]

Last week, akosma software was invited by the immedia[2] team in Durban[3], South Africa, for a week-long workshop about the technical and commercial possibilities that iOS and other mobile platforms offer today. The trip was nothing short of amazing, and provided both immedia and akosma with a great way to exchange ideas, tips, knowledge, and culture. South Africa is an outstanding cocktail of diversity and opportunities, and immedia embraces the new paradigms of technology with imagination, speed and expertise.

One of the founding ideas of akosma software is that the biggest problem in software engineering is human, not technical; the immedia team confirms and embraces this fact, having assembled an awesome multicultural team[4], true to the spirit of South Africa. Even better, Durban is a strategic location of strong business growth, coupled with a dreamy location, on the shores of the Indian Ocean. They have been recognized leaders in their field for the past 10 years, and have a strong portfolio of systems and solutions to showcase[5].

Of course, as with all friendships, the most funny part of our growing relationship is how it all started; as you might remember, in April 2010

[1] http://www.immedia.co.za/
[2] http://www.immedia.co.za/
[3] http://www.durban.gov.za/
[4] http://www.immedia.co.za/gallery/
[5] http://www.immedia.co.za/portfolio/

akosma software brought the first iPads in Switerland[6]. Well, it turns out that Anice Hassim[7] and Kishyr Ramdial[8], respectively founder and technical team leader of immedia, were in the Apple Store line that chilly morning of April 3rd, and they brought the first iPads to South Africa! We later met again in WWDC 2010, when the Football World Cup was starting in their beautiful homeland. Those meetings have evolved into a strong business relationship and a great friendship!

I would like to thank Anice[9], Kishyr[10], Nazeem Ebrahim[11] and all the team for an incredible time. You will hear more about immedia and akosma software working together in the future!

PS: by the way, if your software company is growing and you need strategic guidance, or if you are wondering about the new business opportunities of the mobile market, akosma software is your partner of choice for consulting, business advice, and technical training. We are here to help![12]

[6] https://akosma.com/2010/04/07/first-ipads-in-switzerland/
[7] http://twitter.com/anicehassim
[8] http://twitter.com/kishyr
[9] http://twitter.com/anicehassim
[10] http://twitter.com/kishyr
[11] http://twitter.com/nazeem_ebrahim
[12] https://akosma.com/contact/

Dubai. Babel.

Adrian Kosmaczewski

2011-05-31

Volviendo de Sudafrica con Clau, hicimos escala en Dubai[1], esa ciudad mitica, la que surgió en el desierto en solo 20 años, la del rascacielos mas alto del mundo, y que se yo cuantos superlativos mas. Lo que vimos en 10 horas ahí es algo imperdible.

He aquí la anécdota: la cosa era que nuestra escala era de 11 horas; llegamos de Durban a las 5 de la mañana, y el vuelo para Zurich era a las 4 de la tarde, así que apenas aterrizamos nos buscamos la manera de salir del (inmenso) aeropuerto de Dubai, y conocer un poco la ciudad.

Lo complicado de tal emprendimiento fue que Claudia posee un pasaporte boliviano, el cual requiere visas[2] para entrar a varios países del mundo, incluyendo los Emiratos Arabes Unidos (donde se encuentra, justamente, Dubai). Así que fue medio tanteando la cosa que nos acercamos al mostrador de Emirates (la linea aérea) que nos dijo que, pagando lo que se debe (unos 50 dólares) no hay problema.

Dubai es el reino de la guita, y se nota hasta en estos detalles. Cabe aclarar que Clau no tuvo tal problema en Sudafrica, que es uno de los pocos países de Africa que permite el ingreso de ciudadanos bolivianos sin drama ni visa alguna. Gracias Nelson!

Bueno, para hacer corta la historia, tramite va tramite viene, recién a las 8 llegamos (muertos) al hotel “Marco Polo[3]” de Dubai, en la calle Al Muteena - que a mi me suena como un pariente lejano de la Almudena de Madrid, y seguramente tendrán algo en comun.

Para que se den una idea, en Dubai a las 5 de la mañana la temperatura era de 32 grados. Pasear por la ciudad se convierte en un ejercicio de acrobacia y estilo, entrando y saliendo de lugares con aire acondicionado; son temperaturas que ni los arabes se bancan. Apenas salís a la calle y se te empañan los anteojos de sol. Sentís que tu cuerpo te increpa con un “eeeeeeehhhhh que te paaaaasa???” a cada paso. El pavimento quema, las paredes de cualquier edificio queman, el picaporte de la puerta del taxi quema, todo quema.

[1] http://es.wikipedia.org/wiki/Dubái

[2] http://en.wikipedia.org/wiki/Visa_requirements_for_Bolivian_citizens

[3] http://www.marcopolohotel.net/

Ojo, esto es a fines de mayo, por lo tanto en el hemisferio norte es verano; por ahí el clima es un poquito mas clemente en diciembre, pero algo me dice que no es así.

El gordo Casero habla, en una de sus versiones de "Aquel Maldito Champán", de un calor en la Pampa donde "hasta las lagartijas andaban con sombrilla". Bueno, acá ver gente cruzando la calle con paraguas no es raro. El sol árabe no es moco de pavo. 32 grados, a esa hora, hermano, me hizo recordar algún verano en Buenos Aires con esa temperatura, pero esta vez con muchísima menos humedad (casi nada). Muy sofocante, hasta te cuesta respirar. La sangre polaca en mis venas rechaza tales temperaturas con fervor patriótico, con ahínco y con estupor.

Y con sed. No se olviden la botella de agua mineral si salen a pasear, no es broma lo que les digo. En segundos te deshidratás, mal.

No solo todo es caliente, sino que, además, todo es amarillo; hay arena por todos lados, y los vientos calientes del desierto del Rub' al Khali[4] traen arena continuamente (suena la musica de Laurence de Arabia[5] para acompañar este pasaje de mi relato).

Dubai es una ciudad artificial creada a partir de la nada, en un lapso récord, de solo 20 años. Y se nota; todo esta nuevo, reluciente, las calles son anchas y perpendiculares, es la apoteosis de la ciudad diagramada, una La Plata con esteroides, en la cual las diagonales fueron pensadas como autopistas desde el vamos.

Pero, para ser sinceros, este parece ser el lugar menos indicado del mundo para vivir, o al menos esa es la sensación. Y en medio de este horno, la gente es muy cálida (en todos los sentidos de la palabra, no es chiste); realmente hay una gran amabilidad en el aire, el trato es cordial para con el turista (sobre todo cuando no entendes pepa de arabe, como nos pasa a nosotros y a tantísimos otros). Y cuando digo gente, no digo arabes, porque en Dubai casi casi te diría que son minoría. La gente viene de todos lados; rusos, chinos, srilankeses, hindúes, africanos, todos están aquí.

El ingles es el nuevo latín que, al menos por ahora, mantiene la torre de Babel en pie.

Dubai tiene una actividad económica increíble, y eso se nota de dos maneras: primero, porque hay negocios de cualquier cosa, en cualquier lado, abiertos a toda hora, y por otro lado, porque hay unos atolladeros de transito que no se puede creer. Imagínense que hace 20 años apenas, a esta ciudad la conocía solo el jeque y los 30 pescadores que vivían por ahí. Parece raro, pero no hay manera de evitar atolladeros incluso cuando has puesto autopistas en cada esquina. O mejor dicho, aquello es causa de eso, justamente.

[4] http://es.wikipedia.org/wiki/Rub_al-Jali

[5] http://www.youtube.com/watch?v=irPSvEkQl8Q

Dubai es la ciudad que mejor representa el "Antropoceno[6]", la nueva era geológica de este planeta, en la cual la raza humana se transforma en la mayor fuerza tectónica que haya jamas modificado la faz de la tierra.

Dubai da miedo.

Por ejemplo, se tiene que importar agua para regar los parques (que son hermosos), de otra manera no habría suficiente agua para mantener en vida una flora sorprendente en medio de una desolación absoluta. Porque el desierto no perdona.

Dubai es una lucha perdida de antemano.

Cuando salis de Dubai en dirección de Durban, la ruta del avión atraviesa el famoso desierto arabe que les mencione anteriormente; y créanme, visto desde el avion, es una desolación total. Un mar de arena, no hay nada. Al lado de este desierto, la Patagonia se parece al Amazonas. Al menos hay cardos y tenés alguna probabilidad de perderte y terminar en el Bolsón.

Acá, nada de eso. Te perdiste, sos boleta.

El aeropuerto de Dubai[7] merece otro capitulo. Solamente la terminal 3 constituye en si misma el edificio mas grande del mundo por superficie cubierta[8], tiene dimensiones delirantes, unas 160 canchas de fútbol una al lado de la otra, y encima lo están agrandando; no solo eso, sino que están ya construyendo la terminal 4, como si no hubiese ya bastante espacio para recibir aviones y gente. De los 48 millones de pasajeros que transitan por aquí cada año, creo haber leído por ahí que quieren llegar a los 98 millones de pasajeros. Dos veces Argentina, por año, respirando este aire acondicionado, yendo a cualquier lado del mundo.

Como les digo, el aire adentro de este aeropuerto (como adentro de cualquier taxi, hotel, bar y cabina telefónica en esta ciudad) esta totalmente acondicionado, lo cual nos dejo pensando a Clau y a mi en la cantidad de megawatts por hora que esta gente consume en aire acondicionado. Creo que Itaipú, Atucha y el complejo Chocón-Cerros Colorados juntos no bastarían para sostener tal demanda.

Y finalmente, para seguir con la anécdota, después de una siesta en el hotel, dejamos las valijas y nos fuimos al Burj Khalifa[9], el edificio mas alto del mundo. Visita turística típica y obligada. Buscando la entrada para turistas, en un momento salimos a la calle, y siendo ya mediodía, la temperatura rondaba los 40 grados a la sombra. Como pueden imaginarse, volvimos corriendo a los pasillos del shopping center que esta al lado del edificio. No se banca, así nomas.

[6] http://es.wikipedia.org/wiki/Antropoceno
[7] http://en.wikipedia.org/wiki/Dubai_International_Airport
[8] http://en.wikipedia.org/wiki/List_of_largest_buildings_in_the_world
[9] http://es.wikipedia.org/wiki/Burj_Khalifa

Con respecto al Burj Khalifa, miren, en junio del 2000 pude subir a las torres gemelas del World Trade Center de Nueva York. Unos 110 pisos de alto, una estructura que me dejo atónito en su envergadura. Despues el año pasado subimos con Clau al Empire State.[10] Unos 100 y pico pisos de alto.

Pero el Burj Khalifa, muchacho, esta las manda al vestuario a esas, a las torres Petronas[11] y al Taipei 101[12] todos juntos, con la cola entre las patas y como si el Barcelona las hubiese goleado por 5 a cero. No solamente por su altura, sino por una belleza intrinseca, casi organica.

Me hizo pensar por momentos a los cruceros Calamari[13] que usan los rebeldes para destruir la segunda estrella de la muerte en "El Regreso del Jedi" (no se rían de mis referencias arquitectónicas, por favor). El Burj Khalifa es antisimétrico, altísimo, curvo, parece una planta que emerge del suelo. Es absolutamente delirante. Y la figura de este edificio transforma totalmente el perfil de la ciudad de Dubai cuando la ves desde el aire.

Siguiendo en la linea "Star Wars", Dubai parece como si hubiesen construido Coruscant[14] en Tatooine[15]. El que sea fanático, que comprenda la comparación.

La visita guiada que te hacen en el Burj Khalifa te permite subir solamente (!) hasta el piso 125, desde donde podes ver que la cosa sigue para arriba de manera vertiginosa, imparable. Es mas; desde la terraza de observación para los turistas no se ve la punta, no se ve donde termina el edificio; y obviamente, de eso se trata.

Hay una desmesura en Dubai que supera lo imaginable. No voy a entrar en discursos kitsch sobre la "proyección al mundo", la "demostración de poder", nada de eso; simplemente digo, algo muy loco esta pasando ahí. Les dejo el texto que esta en la entrada, que habla por si mismo.

El texto reza lo siguiente:

> I am the power that lifts the world's head proudly skywards surpassing limits and expectations.
>
> Rising gracefully from the desert and honouring the city with a new glow, I am an extraordinary union of engineering and art, with every detail carefully considered and beautifully crafted.
>
> I am the force of collective aspirations and the aesthetic union of many cultures. I stimulate dreams, stir emotions and awaken creativity.

[10] http://www.youtube.com/watch?v=56PRTsqicvI&feature=channel_video_title
[11] http://en.wikipedia.org/wiki/Petronas_Towers
[12] http://en.wikipedia.org/wiki/Taipei_101
[13] http://en.wikipedia.org/wiki/Mon_Calamari_cruiser
[14] http://en.wikipedia.org/wiki/Coruscant
[15] http://en.wikipedia.org/wiki/Tatooine

> I am the magnet that attracts the wide-eyed tourist, eagerly catching their postcard moment, the centre for the world's finest shopping, dining and entertainment and home for the world's elite.
>
> I am the heart of the city and its people, the marker that defines Emaar's ambition and Dubai's shining dream.
>
> More than just a moment in time, I define moments for future generations.
>
> I am Burj Khalifa.

Escribo estas lineas escuetas mientras sobrevolamos el norte de Irak y Armenia en nuestro vuelo de regreso a casa. Este primer viaje nuestro al hemisferio oriental fue demasiado delirante; tengo los ojos llenos de recuerdos, las papilas con sabores nunca antes imaginados, los oídos con músicas nunca antes vistas. Y con preguntas sin respuesta.

“Introduction to iOS Software Development” Video on InfoQ!

Adrian Kosmaczewski

2011-06-02

The “Introduction to iOS Software Development” session, part of the mobile track at QCon London 2011[1] last March, including the full one-hour long video & slides, is available at the InfoQ site![2]

[3]

By the way, check out the “Mobile App Privacy — You’re Doing It Wrong

[1] http://qconlondon.com/
[2] http://www.infoq.com/presentations/Introduction-to-iOS-Software-Development
[3] http://www.infoq.com/presentations/Introduction-to-iOS-Software-Development

(and So Am I)"[4] session by Graham Lee[5], also available on InfoQ.

Many thanks to the InfoQ[6] team for publishing these talks!

PS: don't forget to check out, from the same conference, Mike Lee's Making Apps That Don't Suck[7] talk!

[4] http://www.infoq.com/presentations/Mobile-App-Privacy
[5] http://twitter.com/iamleeg
[6] http://www.infoq.com/
[7] http://www.infoq.com/presentations/Making-Apps-That-Dont-Suck

Learning One New Language Every Year

Adrian Kosmaczewski

2011-06-04

Here's an update of the current status of my "one language per year" lifelong initiative:

1. 1992: QBasic
2. 1993: Turbo Pascal
3. 1994: C
4. 1995: Delphi
5. 1996: Java
6. 1997: JavaScript
7. 1998: VBScript
8. 1999: Transact-SQL
9. 2000: C# + Prolog
10. 2001: C++
11. 2002: PHP
12. 2003: Objective-C
13. 2004: Visual Basic.NET
14. 2005: Ruby
15. 2006: LINQ
16. 2007: Erlang
17. 2008: Python
18. 2009: Go
19. 2010: Lisp
20. **2011: Haskell**

The trend has roughly been an evolution from procedural during the 90's, to object-oriented ones at the beginning of the 2000's, and finally to functional languages right now.

And thus I realize, I've been programming for 20 years this year, 15 of which for a living.

Foro de Tecnologia en Santa Cruz de la Sierra, Bolivia, el 16 de junio

Adrian Kosmaczewski

2011-06-10

El 16 de junio tendré el privilegio y el honor de exponer a propósito de aplicaciones móviles en el Foro de Tecnologia “24/7 al alcance de tus clientes”[1] que se llevará a cabo en los edificios de la CAINCO en Santa Cruz de la Sierra[2], en Bolivia.

[1] http://www.cainco.org.bo/calendar/Lists/FOROS/DispForm.aspx?ID=38&Source=http%3A%2F%2Fwww%2Ecainco%2Eorg%2Ebo%2Fcalendar%2FLists%2FFOROS%2FAllItems%2Easpx

[2] http://www.cainco.org.bo/

El objetivo del foro es dar a conocer al público, los beneficios y riesgos al exponer la empresa a redes sociales, esto mediante teoría y práctica con casos de empresas y experiencias de éxito. De igual forma presentar las aplicaciones móviles como parte de la estrategia de marketing de las empresas y dar a conocer sus productos a través de aplicaciones descargables para el público en general.

Este foro está dirigido a gerentes de marketing, ejecutivos de tecnología, mercadeo, público en general entendido en el tema, interesados en tener a mano toda la información necesaria para tomar decisiones, y así poder aprovechar las nuevas oportunidades que se presentan actualmente.

Compartiré el podio con José Luis Figueroa[3], consultor y profesor emérito del Tecnológico de Monterrey, cuyos trabajos se enfocan en la estrategia empresarial y la aplicación eficiente de las tecnologías de la información.

Puede bajarse el folleto con el programa completo mediante este en-

[3] http://twitter.com/jlfigueroam

lace[4].

Nos vemos este 16 de junio del 2011 de 8:30 a 12:30 horas en el Centro de Convenciones CAINCO en Santa Cruz de la Sierra! No duden en asistir y espero que el evento sea de gran interés para todos.

[4] http://www.cainco.org.bo/calendar/Lists/FOROS/Attachments/38/programa%20foro%20de%20tecnologia%20jpg.JPG

Why the iPad Is Better Than an Inflight Entertainment System

Adrian Kosmaczewski

2011-06-11

After all my trouble with air travel[1], I thought I should add some positive views here. And they all turned to be around the iPad, so here they go.

The iPad is a better inflight entertainment system because…

- The touchscreen actually works. And when you touch it, you don't disturb the person sleeping in the seat in front of yours.
- It's lightweight.
- The captain cannot interrupt your movie or your picture to tell you some useless facts about the temperature outside or the altitude.
- You get to choose the music and the videos that you want to watch. You should just remember to get them prior to boarding, of course.
- You also get to choose the games you want to play. The choice of games is much larger, and it's called App Store.
- You can even read newspapers, books, magazines, in the same screen. Reading the latest issue of the Economist on my iPad[2] is priceless. It's good to avoid being limited to the "in-flight" magazine provided by the airline ("your free copy!"), which tends to be quite lame, no matter which airline we're talking about.
- You can answer e-mails while you fly (for the moment you cannot sent them, unless you fly in some airline that has a wifi network, and as far as I know, there are only a few with such a feature.)
- You could write a novel in iA Writer[3] or Ommwriter for iPad[4], for that matter, all while you listen to Liszt's "Evening Harmony in D Flat Major"[5]. Or you could prepare a blog post, like this one.
- Coupled with noise-cancelling headphones[6], the quality of sound is years-light ahead of what those crummy airline headphones

[1] /blog/i-hate-you-airline-industry/

[2] http://www.economist.com/digital/apps

[3] http://www.informationarchitects.jp/en/writer-for-ipad/

[4] http://www.ommwriter.com/

[5] http://twitter.com/#!/akosma/statuses/27926124027

[6] http://www.bose.com/controller?url=/shop_online/headphones/noise_cancelling_headphones/index.jsp

are able to provide.

- The battery. A whole 10-hour flight on a single charge is absolutely possible.

'Nuff said.

Resumen del Foro de Tecnología de CAINCO

Adrian Kosmaczewski

2011-06-19

(Para el comunicado de prensa original, siga este link[1])

Foro de tecnología de CAINCO:

Las redes sociales llegaron para quedarse

El profesor del Tecnológico de Monterrey-México, José Luis Figueroa, junto a Adrián Kosmaczewski, desarrollador de aplicaciones para iPhone, Mac y especialista en iPad, disertaron en el Foro de Tecnología: Aplicaciones Móviles y Redes Sociales, evento organizado por la Cámara de Industria, Comercio, Servicios y Turismo de Santa Cruz (CAINCO), en el marco del programa AL-Invest IV.

[2]

“Si alguno aún no tiene presencia en alguna de las redes sociales, lo invito a que lo haga, porque quizás sus competidores ya las están aprovechando”, dijo el profesor del Tecnológico de Monterrey-México, José Luis Figueroa, quien junto a Adrián Kosmaczewski, desarrollador de aplicaciones para iphone y iPad, disertaron en el Foro de Tecnología: Aplicaciones Móviles y Redes Sociales, evento organizado por la Cámara de Industria, Comercio, Servicios y Turismo de Santa Cruz (CAINCO), en el marco del programa AL-Invest IV.

Los más de 150 asistentes pudieron conocer de cerca los beneficios de las redes sociales para aplicarlos en sus empresas, además de la utilización correcta de las mismas como herramientas de marketing.

De acuerdo al estudio publicado por silicon.com (Shelley Portet) en abril del 2011, el número de cuentas en redes sociales es mayor a la población mundial. En el mismo artículo se comenta que de ellas, cerca de 4.500 millones de esas cuentas están activas. Su crecimiento se debe a la inclusión de videojuegos en algunas redes como

[1] http://www.cainco.org.bo/salaPrensa/notasPrensa/Notas%20de%20Prensa/Nota_Foro_Tecnolog%C3%ADa_Resumen.doc

[2] http://www.cainco.org.bo/salaPrensa/notasPrensa/Notas%20de%20Prensa/Nota_Foro_Tecnolog%C3%ADa_Resumen.doc

Facebook, y el hecho que los dispositivos móviles son cada vez más inteligentes y aceptados por la población, como las “tablets” (iPad) y los “smartphones”, (teléfonos inteligentes), ha hecho que el acceso a las redes se haga desde cualquier parte donde se encuentre el usuario.

“Las redes sociales, hoy apoyan los procesos de comunicación interna y externa, con clientes, proveedores, empleados y público en general. Utilícelas”, remarcó Figueroa.

Ante esta realidad, Adrián Kosmaczewski, expuso sobre la importancia y la evolución tecnológica que hemos vivido en los últimos tiempos, desde la llegada del teléfono hasta la actual revolución de los “smartphones”.

“Existen 94 millones de smartphones actualmente con un alto crecimiento anual. Según las encuestas, en el 2012 existirán más smartphones que pc’s.”, señaló Kosmaczewski.

El expositor comentó sobre su experiencia en el desarrollo de aplicaciones y la importancia de las mismas para poder llegar de manera más rápida y directa a los clientes, haciendo énfasis en las ventas de las mismas a través de la tienda virtual App Store de Apple.

“No tengan miedo a ser criticados por los malos comentarios o errores que noten los clientes en las aplicaciones que desarrollen, ya que esto puede ayudarnos a mejorar nuestros productos y hacerlos más eficaces con nuevas versiones de los mismos”, aseguró.

Para mayor información comunicarse con Gabriel Columba cel 76612275 Ejecutivo Principal – DIRCOM CAINCO E-mail: gabriel.columba@cainco.org.bo[3]

Puede obtener también la nota completa ingresando a la página de Cainco www.cainco.org.bo[4]

[3] mailto:gabriel.columba@cainco.org.bo
[4] http://www.cainco.org.bo/

Article in the SonntagsZeitung: “Die neuen Schweizer Macher”

Adrian Kosmaczewski

2011-06-19

The SonntagsZeitung, the most widely read Sunday newspaper in Switzerland, features today an article + infographic about Swiss developers making mobile applications, and akosma software[1] is featured (among many other friends and peers) together with the swissinfo iPad app[2]!

[3]

[1] http://akosma.com/
[2] http://www.swissinfo.ch/eng/services/ipad.html?cid=28928468
[3] /press/SonntagsZeitung-Swiss-Made-Apps.pdf

Pais Central

Adrian Kosmaczewski

2011-06-20

Hace mucho tiempo, Luisa, una amiga de toda la vida, me hizo una de esas preguntas que te dejan en offside y quedan picando durante largo rato: “que se siente vivir en un país central?”

Para que sepan, Luisa es socióloga, de las del alma. Quedan pocas como ella; son una especie rara que no se si esta en riesgo de extinción, pero algo es seguro; ninguna pregunta de ella es anodina.

Recuerdo aun que me agarró totalmente desprevenido. Nunca pensé en Suiza como un país central; es mas, para ser sincero, siempre me pareció que estaba bastante alejado del centro (cual sea el mismo). Particularmente en mi industria, la informática, siempre me pareció que el centro era en realidad Silicon Valley, y por extensión los Estados Unidos. El adjetivo mas común que empleo al hablar de Suiza suele ser “Tupperware”. Pero un “país central”? No.

Y sin embargo, pensandolo bien, ahora creo que quizás este Tupperware este en el centro del centro, que sea el núcleo mismo. El vivir en Argentina durante unos años me permitió ver eso de lejos; efectivamente Suiza es un país central. Y flor de país central.

Pero, que significa eso?

En un país central, las cosas circulan por él; como esta en el centro, esta en medio de todos los caminos. Las rutas son mas cortas pasando por ahí, tanto en el sentido propio como figurado de la expresión.

En un pais central, confluye la gente de todo el mundo; se hablan todos los idiomas, se consiguen todas las especias y se mezclan todas las razas. Se prueban todas las gastronomías, se bailan todas las danzas.

En un pais central, todo se ve como anillos concéntricos que rodean ese centro; y precisamente, es exactamente así como la diplomacia Suiza caracteriza al resto del mundo. Basicamente, con Suiza en el centro, Europa es el primer anillo. Luego vienen los Estados Unidos, Canada, Australia, Japon y Nueva Zelandia, y mas afuera los países del medio oriente, Brasil, Rusia, India y China. Finalmente, en el anillo mas excéntrico, los demás países de Latinoamérica, Africa y Asia. No es broma; esto me lo contó un diplomático.

En un pais central, el comercio no se detiene nunca, y es siempre de envergadura; quizas no se vea a nivel del dia a dia, pero la verdad es que en inmensas cantidades, todos los bienes del mundo circulan por un pais central, tarde o temprano. Hay que saber que la mayor parte del petróleo mundial se negocia, se vende y se compra, cada dia, en Ginebra[1] (de manera virtual, claro está; nadie quiere un Panamax[2] en el lago). Y obviamente, no solamente se comercia petróleo.

En un pais central, las cosas no necesitan ser demasiado grandes ni pequeñas, ni demasiado simples ni demasiado complejas; ya que todo circula, todo se consigue, todo se mide. El arte y la arquitectura Suizos son tipicamente minimalistas y funcionales. Las consideraciones estéticas se han dejado de lado hace rato, o tomaron radicalmente otro sentido.

En un país céntrico no hay apuro, no hay prisa, no hay tiempo. El tiempo se construye precisamente ahí. En una ciudad llamada La Chaux-de-Fonds[3], para ser mas preciso, donde la industria relojera fue declarada patrimonio mundial intangible de la humanidad, y donde casi casi hay mas marcas de relojes que habitantes.

En un pais central, todo se mueve, y sin embargo todo se queda. El sector primario sigue siendo una actividad central, y los agricultores una raza protegida; la cantidad de subvenciones para mantener el sector agricola superan ampliamente el presupuesto de pequeños países. Pero claro, que seria del turismo suizo si no estuviese la vaca Milka.

En un pais central, en resumidas cuentas, la desmesura es invisible y discreta. Un Porsche es un auto común y corriente, un taxi que se digne debe ser un Mercedes, un tren tiene que tener primera clase, el ruido no se tolera, las vacas tienen prerrogativas diplomáticas y todo lo que viene de afuera es, a priori, de temer.

Por todo esto, Luisa, tenias razón; Suiza es un país central. Es mas, quizás sea el país central por antonomasia.

Que se siente vivir en el? Pucha, me olvide de contestarte la pregunta una vez más. Aunque ya escribí varios posts en este blog[4] hablando de eso, así que te dejo leer y comentar :)

[1] http://www.nowpublic.com/tech-biz/geneva-world-oil-trading-capital
[2] http://es.wikipedia.org/wiki/Panamax
[3] http://es.wikipedia.org/wiki/La_Chaux-de-Fonds
[4] /tags/switzerland/

Interview in the Bolivian TV

Adrian Kosmaczewski

2011-06-30

In the context of the event organized by CAINCO[1] in Santa Cruz de la Sierra, Bolivia, on Thursday, June 16th, we were invited to be interviewed in the TV program "No Mentirás"[2] ("You will not lie") of the PAT network[3], in the evening of Wednesday 15th. The program is one of the most watched evening news shows in the country, and the objective was to raise awareness about the event to be held the following day.

The videos are available now on Justin.TV[4] with the whole contents of the interview (in Spanish): part 1[5] and part 2[6]

Here go some photos taken from the videos:

[1] http://www.cainco.org.bo/
[2] http://www.redpat.tv/nomentiras/
[3] http://www.redpat.tv/
[4] http://www.justin.tv/
[5] http://www.justin.tv/patscz/b/288186366
[6] http://www.justin.tv/patscz/b/288186417

From left to right: Jose Luis Figueroa, Adrian Kosmaczewski, Olivia Stelzer, and Ezequiel Serres, host of the program.

More photos:

Olivia Stelzer, Commercial Executive at CAINCO

Jose Luis Figueroa and Adrian Kosmaczewski

Jose Luis Figueroa

Adrian Kosmaczewski (no comments on the orthography of the family name :))

Our interview begins in the video below[7], at around the 1h 29min mark:

Watch live video from PAT desde Sta Cruz on Justin.tv[8]

The second part of the interview is in this second video[9]:

Watch live video from PAT desde Sta Cruz on Justin.tv[10]

[7] http://www.justin.tv/patscz/b/288186366
[8] http://www.justin.tv/patscz#r=-rid-&s=em
[9] http://www.justin.tv/patscz/b/288186417
[10] http://www.justin.tv/patscz#r=-rid-&s=em

New Visual Identity

Adrian Kosmaczewski

2011-07-02

We are thrilled to unveil the new visual identity of akosma software, a pure product of the incredible technique and art of moser[1]!

We would like to thank all the moser team, in particular Emilie Mattille, Sophia Delacrétaz and Alexandre Henriques, for their incredible skills and patience in the creation of an outstanding visual identity. It represents in a simple form the values, the beliefs, our way of doing business, but also our imagination and a continuous search for innovation.

Of course, the logo is just one part of it, albeit a very important one. More will be shown very soon. Stay tuned!

[1] http://www.moserdesign.ch/


AKO
SMA
AKO
SMA
AKO
SMA
AKO
SMA

Student application on the App Store: Sudokulus

Adrian Kosmaczewski

2011-07-06

Remember the advanced training session[1] I've given back in February? Well, today I've had the nicest of surprises when Pieter Muller, one of the students that attended it, sent me the link (and a complimentary promo code!) to his very first app in the App Store!

[2]

Here it is; it is called Sudokulus[3], and it is an amazing app that literally solves any Sudoku puzzle you point the camera at, automatically, in a snap. You have to see it to believe it! I've tried on printed and even on a Sudoku on my iPad (with the iPhone pointing at it) and it just works. Awesome!

Congratulations Pieter! Let's give his app some download love!

[1] /blog/advanced-ios-4-2-training-course-zurich-february-7th-and-8th-2011-enroll-now/

[2] http://itunes.apple.com/ch/app/sudokulus/id443267921?mt=8

[3] http://itunes.apple.com/ch/app/sudokulus/id443267921?mt=8

[4]

[4] http://itunes.apple.com/ch/app/sudokulus/id443267921?mt=8

Useful 3rd Party Extensions to CoreTextWrapper

Adrian Kosmaczewski

2011-07-07

Our CoreTextWrapper project in Github[1] is certainly popular! Many developers have told us that they find it easier to approach Core Text using it, providing an easy learning path, showing the key concepts with an easy-to-use example.

Some developers have gone the extra mile, though: we have kindly received two enhancements for the project!

- Jared Crawford[2] contributed support for shadows in AKOCustomFontLabel back in March.[3]
- Today, Christian Menschel from tapwork.de[4] contributed the AKOMultiColumnTextViewDataSource protocol to add custom views in the text columns.

These contributions make the CoreTextWrapper a very interesting option to display rich, multi-column text on your iPad application! Thanks to Jared and Christian for the contributions, and don't hesitate to send us your pull requests or your changes for us to integrate with the project. We would love to make your code part of this project.

[1] https://github.com/akosma/CoreTextWrapper

[2] https://github.com/JaredCrawford

[3] https://github.com/akosma/CoreTextWrapper/commit/66af5409db46410e3cc8f5490ea9f4418e2843be

[4] http://www.tapwork.de/


News of the Day
Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in
culpa qui officia deserunt mollit anim id est laborum.
Sed ut perspiciatis unde omnis iste natus error sit voluptatem accusantium doloremque laudantium, totam rem aperiam, eaque
ipsa quae ab illo inventore veritatis et quasi architecto beatae vitae dicta sunt explicabo. Nemo enim ipsam voluptatem quia voluptas sit aspernatur aut odit aut fugit, sed quia consequuntur magni dolores eos qui ratione voluptatem sequi nesciunt. Neque porro quisquam est, qui dolorem ipsum quia dolor sit amet, consectetur, adipisci velit, sed quia non numquam eius modi tempora incidunt ut
News of the Day
Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do
voluptatem quia voluptas sit aspernatur aut odit aut fugit, sed quia consequuntur magni dolores eos qui ratione voluptatem sequi nesciunt. Neque porro quisquam est, qui dolorem ipsum quia dolor sit amet, consectetur, adipisci velit, sed quia non numquam eius modi tempora incidunt ut labore et dolore magnam aliquam quaerat voluptatem. Ut enim ad minima veniam, quis nostrum exercitationem ullam corporis suscipit laboriosam, nisi ut aliquid

More about our new graphic identity

Adrian Kosmaczewski

2011-07-22

The great work of >moser[1] is also proudly displayed in the graphic work they've done for us, far beyond the logo; check out these images, and also a surprise at the end of this post!

[1] http://www.moserdesign.ch/


AKO
SMA


AKO
SMA
AKO
SMA
AKO
SMA

So now get your brand new akosma gear, ready to be ordered through the akosma shop[2]!

[3]

[2] http://akos.ma/shop
[3] http://akos.ma/shop

In the picture above: coffee mugs[4], a meeting notebook[5], stickers[6], and the akosma iPhone 4 Slider Case[7]!

[4] http://www.cafepress.co.uk/akosma.552053458
[5] http://www.cafepress.co.uk/akosma.552053448
[6] http://www.cafepress.co.uk/akosma.552053462
[7] http://www.cafepress.co.uk/akosma.552053446

Markdown FTW

Adrian Kosmaczewski

2011-07-31

Markdown[1] is my new favorite tool.

It all started while looking for alternatives to LaTeX[2] to write documents and booklets, because since the release of the iPad last year, I wanted to publish in PDF and in EPUB format at the same time, and LaTeX does not offer that option off the box.

And, besides, I **really** found that LaTeX was a great system, but reading LaTeX code was not always enjoyable. It's a bit of a messy markup language.

So that's how I learnt about Pandoc[3]; it is an incredible tool written in Haskell that pretty much transforms any kind of markup into another: RTF, MediaWiki syntax, HTML, LaTeX, Textile, you name it. However, it extends and gives a special status to Markdown[4], as one of its primary formats, and it provides support to create PDF and EPUB files out of the box; bingo, that's exactly what I needed. Even better, it uses LaTeX to generate PDF files, which means that I can reuse my LaTeX knowledge to generate beautiful documents.

But then, learning more about Markdown[5] (the syntax is not very far away from Textile[6], which I knew better), I remembered that StackOverflow[7] uses it; that GitHub[8] uses it; and then I found an excellent Markdown plugin for WordPress[9] and another great Markdown Redmine plugin[10]. Then I updated Elements[11] recently on my iPad (an excellent Dropbox-powered text editor for the iPad), and found out that it had native Markdown[12] support. And of course, both MacVim[13]

[1] http://daringfireball.net/projects/markdown/
[2] http://www.latex-project.org/
[3] http://johnmacfarlane.net/pandoc/
[4] http://daringfireball.net/projects/markdown/
[5] http://daringfireball.net/projects/markdown/
[6] http://www.textism.com/tools/textile/
[7] http://stackoverflow.com/
[8] https://github.com/
[9] http://wordpress.org/extend/plugins/markdown-for-wordpress-and-bbpress/
[10] https://github.com/juno/redmine_markdown_extra_formatter/tree
[11] http://www.secondgearsoftware.com/elements/
[12] http://daringfireball.net/projects/markdown/
[13] http://code.google.com/p/macvim/

and TextMate[14] have an excellent Markdown[15] support, including syntax highlighting and preview. And finally, even better, I discovered that MarsEdit supports Markdown natively[16].

So that's it, I'm sold. I'm writing almost everything these days with Markdown[17]. And it's a simple, pure joy.

PS: I'm even considering buying Marked[18] by Brett Terpstra, or even Macchiato[19] (although this last one seems to me a bit pricey).

[14] http://macromates.com/
[15] http://daringfireball.net/projects/markdown/
[16] http://www.red-sweater.com/marsedit/
[17] http://daringfireball.net/projects/markdown/
[18] http://markedapp.com/
[19] http://getmacchiato.com/

Entrevista en la Metro 95.1 esta noche!

Adrian Kosmaczewski

2011-08-05

Esta noche me van a entrevistar en el programa “Su Atención Por Favor”[1] de la radio Metro 95.1[2] de Buenos Aires. Hay una sección de argentinos en el extranjero y bueno, hoy me toca a mi desde Suiza :) Será alrededor de las 22 de Buenos Aires (algo así como las 3 de la mañana del sábado por acá! Voy a estar carburando al café para no quedarme dormido ja)

Mil gracias a mi querida amiga de toda la vida Betina Suárez[3], que fue la que permitió el primer contacto con la gente de la radio para que esto suceda! No se pierdan su blog “Mujer, Madre y Argentina”[4], absolutamente delirante.

PS: acabo de ver que la Metro 95.1 tiene una app iPhone[5]; bajando!

[1] http://www.metro951.com/index.php?option=com_programas&view=programas&id=42

[2] http://www.metro951.com/

[3] https://twitter.com/#!/MMyArgentina

[4] http://mujermadreargentina.com.ar/blog/

[5] http://itunes.apple.com/au/app/metro-95.1/id410630212?mt=8

Why do not we outsource projects overseas?

Adrian Kosmaczewski

2011-08-05

Given the number of times we have been asked about this subject, and the puzzled looks after we give an answer, here goes an official statement:

akosma software does not outsource its projects.

Against the common opinion of most IT newspapers and blogs, we think there are several reasons to not doing it:

1. Outsourcing is a modern variant of that old north-south dependency relationship. In the XIXth century it was about cotton, coal or food. These days is about software. But the schema is the same; the north buys cheap goods and services from the south, but the added value remains always in the north. We believe that **this state of things is one of the main reasons of the current state of the world**; and thus, we prefer to teach the south how to create their own value, rather than selling it to companies who will, in turn, build a name upon your own effort. We think that **talking about sustainable development** also means refraining from outsourcing.
2. Making software is our core business. We do not think that outsourcing our core business is a good idea. We have seen many software companies struggle with the management of outsourced projects, and we do not want that. We do not care if you are agile, waterfallish or CMMI level 42.
3. We work in a close relationship with our clients; we seek their input several times a day, we ask for their feedback continuously, and we insist in having them participate in the development of their products. Working in an outsourcing schema adds a third layer of communication in the whole process, and we do not work well in such situations.

Are we losing money in the process? Sure; but that money would be built upon the effort of others than us. We believe in (and work with) networks of independent peers, providing value at each step of the way: designers, user experience experts, marketing people, creative teams. Not remote companies where the only driving mechanism for the whole relationship is a mere economic advantage.

At akosma software we do not work **for** others, neither want others to work **for** us: we work **with** others, both clients and suppliers, in equal terms. The final product is a joint effort of equal commons. That's how we see things.

Of course, this is our policy, and your mileage and opinions may differ from ours. Maybe outsourcing works in your case; well, good for you, but it definitely does not fit our values, that's all.

PS: message to all outsourcing companies and offshoring firms anywhere in the planet; you can stop contacting us (phone, email, etc). We are not going to work with you, for the reasons above. We think you should probably start developing your own products and services rather than selling them to Europe or the USA.

Votar en Suiza

Adrian Kosmaczewski

2011-08-14

Como muchas cosas en la vida, el pasaporte colorado no viene gratuitamente. Una de esas cosas que hacen a la vida suiza es la obligación de hacer la colimba (o de pagar el impuesto militar si no sos apto para el servicio). La otra es la capacidad de votar. De la colimba, ya escribiré otra vez, pero esta vez hablaremos del voto.

En resumidas cuentas, el voto en suiza tiene las características siguientes:

- **El voto no es obligatorio.** La tasa de participación oscila entre el 30%, un promedio bastante común, y el 60%, en el caso de las votaciones con mayor interés. Generalmente son los temas de impuestos y del sistema jubilatorio los que generan mayor participación.
- **El voto femenino**[1] **se instauró a nivel federal recién en 1971;** algunos cantones permitieron el voto a las mujeres desde los años 50, pero algún que otro cantón lo siguió prohibiendo hasta los ’90. Obviamente, como en Suiza se vota todo, también se votó para este tema, y el electorado masculino votó en contra del tema varias veces.
- **Se vota cuatro veces por año, casi siempre más o menos en las mismas fechas.** En cada fecha se mezclan temas federales, cantonales y comunales, que muchas veces no tienen nada que ver entre si. No es obligatorio votar en todos los temas que se proponen; se puede cortar boleta y solo opinar sobre un tema de los propuestos.
- **La mayoría de las votaciones son referéndums,** donde se vota por un si o por un no. En las boletas se escribe a mano la respuesta (“Oui” o “Non”, “Ja” o “Nein”) y listo. Generalmente, cada tema de referéndum viene con su contraproposición, elaborada por los partidos opositores al tema del que se vota, y al votar se puede elegir el tema, la contraproposición, y de emitir una opinión en caso de empate (lo cual ha sucedido).
- **Se puede votar por correo.** Unas 3 o 4 semanas antes de la fecha del voto, te llega un sobre con las boletas de voto y las instrucciones de como llenar los formularios, y, aún mas importante, toda la documentación que explica lo que se vota. El voto

[1] http://www.swissworld.org/es/poblacion/mujeres/sufragio_femenino/

por correo es digno de otro post!

- Generalmente las votaciones implican una **modificación de algún artículo de la constitución.** En este sentido, la constitución suiza es mucho mas dinámica que la argentina, y el procedimiento para modificar artículos es mucho mas simple y directo. En realidad, la constitución suiza se parece más a una extensión del código civil, con un valor judicial extendido y un nivel de detalle bastante espeluznante.
- **El resultado de las votaciones tiene generalmente valor de promulgación;** una ley que recibe un voto afirmativo entra en vigor casi inmediatamente.

Es una lástima que me olvidé de sacarle fotos a las boletas que me llegaron para las próximas votaciones, las del 4 de setiembre próximo. Será para otro post!

Speaking at the Mobile Developer Summit in Bangalore

Adrian Kosmaczewski

2011-08-15

I am thrilled to announce that I will be speaking at the Mobile Developer Summit[1] in Bangalore, India, on November 2-3! This event, organized by Saltmarch Media[2] will be hosted at the premises of Indian Institute of Science.

My schedule includes the following talks:

- Write your First iPhone App
- Accessing Web Services from iPhone and iPad Applications
- Mobile Web Rising
- Ten Commandments for iPhone Development

Sharing the stage with Noah Gift, Brian LeRoux (of PhoneGap fame) and Jonathan Saggau, all in the heart of one of the most important technology centers of Asia is an incredible honor. I hope to see you there in November!

[3]

[1] http://www.developermarch.com/mods/
[2] http://www.saltmarch.com/
[3] http://www.developermarch.com/mods/

Dropping support for iPhone OS 3.x

Adrian Kosmaczewski

2011-08-23

Given our strong focus on iOS, and also to be sure to be economically competitive, we are continuously watching the market to know which OS version is used by the many potential users of the apps we make.

Today, we announce that **we officially drop support for iPhone OS 3.x in all of our future projects**. We are not going to accept any more projects with the requirement of compatibility with 3.x or previous versions of iOS (formerly know as iPhone OS), for any device.

There are several reasons to justify this policy:

- There have been many fundamental, non-backwards compatible API changes between iPhone OS 3 and iOS 4, for example the introduction of Objective-C blocks and Grand Central Dispatch, changes in the MediaPlayer framework, in Core Animation, in Core Text, etc.
- Apple has already marked as deprecated or "non recommended" many APIs throughout the frameworks. These will most probably not be supported in future versions of iOS 5.
- Keeping backwards compatibility, given the changes enumerated above, would imply writing, testing, supporting and maintaining a relatively larger code base taking this fact into account.
- Finally, it also implies keeping older devices in order to test applications in them, which leads to harder and costlier management. We need those devices for testing upcoming applications in beta versions of iOS 5!

Furthermore, many statistics show that the proportion of iOS devices still running iPhone OS 3.x is quickly dropping below the 2% mark. That makes it not sustainable nor reasonable to support those devices. For example (click on the links below for the sources):

- In July 2010, after just one month, iOS 4.0 users already represented 50%[1] of all iOS users.
- In December 2010 and January 2011, several[2] statistics[3] were already showing that almost 90% of users were running iOS 4.x.

[1] http://insights.chitika.com/2010/ios-4-now-powering-50-of-iphone-traffic/

[2] http://www.quora.com/What-proportion-of-all-iPhone-owners-use-iOS4-*-today

[3] http://www.readwriteweb.com/mobile/2011/01/what-percentage-of-iphone-owners-are-on-ios4.php

- In March 2011, this trend was confirmed[4] by even more reports[5].
- In June 2011, the proportion of iPhone 3.x users had dropped to 6%.[6]
- And finally, the latest reports[7] for August[8] show that less than 2% of iOS users are stuck with the 3.x version in their devices.

Keeping backwards compatibility in our applications to iPhone OS 3.x isn't justified, then, neither economically nor technically. The rate of upgrade to new versions of iOS is surprisingly high and fast, compared to other mobile platforms.

[4] http://insights.chitika.com/2011/ios-update-ipads-iphones-running-high-rate-of-ios-4/

[5] http://blog.jcmultimedia.com.au/2011/03/is-it-worth-supporting-ios-3-in-2011.html

[6] http://insights.chitika.com/2011/just-in-time-for-ios-5-ios-3-almost-dead/

[7] http://www.marco.org/2011/08/13/instapaper-ios-device-and-version-stats-update

[8] http://www.cocoanetics.com/2011/08/ios-versions-in-the-wild/

Audio de la Entrevista en la Metro 95.1

Adrian Kosmaczewski

2011-09-05

Para los que no pudieron escuchar la entrevista[1] que me hicieron en la emisión “Su Atención Por Favor”[2]. Creo que es la secuencia más larga de clichés que se pueden decir de Suiza en unos 14 minutos :)

Es más, hay algunos datos que no son totalmente correctos (como lo de que no hay presidente, cuando en realidad si, pero no cumple el mismo rol que en Argentina), pero bueno, Sherlock Holmes tampoco nunca dijo “Elemental mi querido Watson” :)

El audio, en formato MP3, grabado mediante Snowtape 2[3], esta a vuestra disposición en Dropbox![4]

[1] /blog/entrevista-en-la-metro-95-1-esta-noche/

[2] http://www.metro951.com/index.php?option=com_programas&view=programas&id=42

[3] http://www.vemedio.com/products/snowtape

[4] http://dl.dropbox.com/u/1451337/entrevista.zip

A Shift In The Market Towards Mobile Web Apps

Adrian Kosmaczewski

2011-09-13

Mobile apps are hardly something new these days. Mobile app stores are ubiquitous, and download and "number of apps" stats do not matter anymore. "Mobile First" is now, more than just a guideline, simply the way business work.

In Switzerland, the big craze of mobile apps began with the introduction of the iPhone in June 2008, and then the waves of new Android devices that have been introduced in the past 2 years. Everybody has a smartphone these days, and now everyone is jumping to buy a tablet (usually an iPad) as fast as they can.

But the mobile market is **highly fragmented**. I'm not talking about the Android hardware market, which is inherently fragmented by nature; we are talking about the app market in general. Companies want to have their services and products in as many smartphone platforms as possible these days, and the iPhone is no longer enough.

To put things in context, the latest statistics[1] of smartphone platforms roughly show that:

- 40% of the market is held by Android devices, with a large proportion of Samsung devices, and apparently with some signs of starvation;
- 30% by iOS, with a small growing trend; not taking into account the iPod touch and the iPad, which are iOS devices, but not really smartphones; and
- 20% by BlackBerry devices, although this proportion is actually on a negative slope these days.

If we consider the "mobile market" as the sum of smartphones and tablets, and if we include the iPod touch into the mix, then iOS holds more than 40% of the whole mobile market. Impressive, but still, not even half of the market.

Given this split, companies want to **maximize the return of their investments**. In other words, they want more bang for their buck. They are still going to spend money for mobile applications, for sure;

[1] http://www.asymco.com/category/industry/

but they are not so sure these days about the “iPhone only” or even “iPhone first” approach that was common until now; **they want an app that works in as many platforms as possible**. Certainly the grim economic situation also plays a role in these decisions!

This is a reality; this is something that we have been experiencing for the past months, and the trend is going in this direction. **2012 is going to be the year of the mobile cross-platform app**, but because the market said so, not because some technology fashionista decided it, so we need to prepare to bring this vision into reality.

In our case, at akosma software we have chosen a few technologies to bet our company on:

- The whole HTML5 standard[2], including the amazing suite of tools that the new generation of browsers are making available for us!
- Sencha Touch[3], the mobile evolution of the Ext JavaScript framework of yesteryear, of which Adrian has been a great fan for years![4]
- jQuery Mobile[5], the promising new framework based on the non less amazing jQuery[6] project!
- PhoneGap[7], the open source mobile web app packaging framework now becoming a de-facto standard!

Are we dropping the iOS platform for web technologies? **Not at all;** we are still going to provide iOS application development services[8], simply because (at least right now) not everything can be done with web apps; but we are proud to add Sencha, jQuery Mobile and PhoneGap as part of our formal offering, because we firmly believe in the huge potential of these platforms.

Moreover, some clients still want to provide custom-tailored experiences to iOS devices, and we fully support those initiatives with our skills and imagination!

In the short term, watch out for this:

- Adrian is going to talk[9] about web apps from an iOS developer perspective at a SkillsMatter[10] event in London, on November 22nd.
- We will be attending SenchaCon 2011[11] in Austin TX (USA), from October 23rd to the 26th, to learn more about Sencha and the next generation of web technologies.

[2] http://mobilehtml5.org/
[3] http://www.sencha.com/
[4] /blog/ext/
[5] http://jquerymobile.com/
[6] http://jquery.com/
[7] http://www.phonegap.com/
[8] http://akosma.com/services/
[9] http://skillsmatter.com/podcast/os-mobile-server/native-cross-platform/js-2661
[10] http://skillsmatter.com/
[11] http://www.sencha.com/senchacon2011

- We will be presenting in the next Mobile Developer Summit[12] in Bangalore, India, in November 2nd.

In the next few months, expect more around web technologies from akosma software!

[12] http://www.developermarch.com/mods/

bluewoki 2.0 Hits the App Store!

Adrian Kosmaczewski

2011-09-15

Version 2.0 of our flagship application bluewoki[1] has been approved (in less than half a day!) and is already available at the App Store! If you haven't heard of it yet, bluewoki is a peer to peer walkie-talkie for iPhone and iPod touch (it also works on the iPad, of course, but with the iPhone look and feel of course).

What is new this new version?

- We have added support for connecting through wifi networks, which effectively turns bluewoki into a nice communication device for your home; just select the "online" option when connecting and select the device you want to talk to, and you are done!
- Also, we have added support for multitasking; you can now hit the home button while talking, go browse the web or launch any other application, and the call stays on! (of course this only works on iOS devices like the iPhone 4, which have support for background tasks).
- Finally, we have upgraded our graphics to support the gorgeous Retina display.

The source code of bluewoki is, as always, available on Github[2]; in

[1] http://bluewoki.com

[2] https://github.com/akosma/bluewoki/

particular, this version is built on top of the great AsyncSocket[3] project to provide wifi connectivity, and might be a good example (we hope) of how to use the chat capabilities of GameKit using a standard wifi network.

We hope you will like this new version! Get bluewoki now on the App Store[4] (available only for devices running iOS 4.0 or higher) or clone the source code on Github[5], and enjoy!

[3] http://code.google.com/p/cocoaasyncsocket/
[4] http://itunes.com/apps/bluewoki
[5] https://github.com/akosma/bluewoki/

A Proposed Architecture for Network-Bound iOS Apps

Adrian Kosmaczewski

2011-09-20

One of my most popular answers[1] in StackOverflow is the one I gave to the following question: "What is the best architecture for an iOS application that makes many network requests?"

The problem is the following: let's consider a relatively complex application, which has to connect to, and retrieve and send data from different remote resources (from the same origin or from different ones), all while handling as gracefully as possible different problems such as "network not available", "500" errors, etc, while at the same time notifying the app about showing popups, enabling and disabling fields, with many different screens (usually each with its own controller), and so on.

This article will describe in detail a solution for this problem using ASIHTTPRequest[2], my favorite HTTP library for iOS. The solution involves a bit of object oriented code, and there is a sample implementation in our Senbei project in Github[3] that I am going to refer to in this article.

[1] http://stackoverflow.com/questions/4810289/best-architecture-for-an-ios-application-that-makes-many-network-requests/4823001#4823001

[2] http://allseeing-i.com/ASIHTTPRequest/

[3] https://github.com/akosmasoftware/Senbei/

I want to point out that I do not consider this the “best” architecture by any means; it is simply a pattern or structure that has given me excellent results in many different applications, and which has evolved over time from many other approaches. If someone else has a better idea, I’d be glad to try it! This architecture also has the advantage of being easy to document, understand, maintain and extend.In the answer on StackOverflow I just enumerated the different elements of the architecture; here I will explain the rationale behind every decision.

Singleton, “Class Cluster”

As I said in the answer, this architecture involves a single object taking care of network connectivity, which I will call a “network manager”. Typically this object is a singleton (created using Matt Gallagher’s Cocoa singleton macro[4]). Basically this is because it’s a good way to centralize all the network logic in a single component, and it is also a very common Cocoa design pattern.

This object can also be seen as a class cluster, because it uses an army of individual classes that perform the real work behind the scenes.

In Senbei, this singleton object is the SBNetworkManager[5] class. All the controllers of the application use the methods of this class to trigger asynchronous requests to the remote FFCRM server used by the application. All of these controllers, as well as the application delegate, are notified of events by means of ad hoc notifications (defined in the SBNotifications.h[6] file).

Network Queues

The network manager wraps an instance of ASINetworkQueue, and also acts as its delegate. Network queues are interesting in mobile apps, given that the available bandwidth varies drastically when the device is connected through a wifi connection or a low-speed GPRS mobile network. The network queue will automatically change the number of requests sent by unit of time depending on the current connectivity, without clogging the device.

In our example, SBNetworkManager has a private ivar (well, as private as Objective-C allows ivars to be) pointing to an instance of ASINetworkQueue, itself a subclass of NSOperationQueue.

[4] http://cocoawithlove.com/2008/11/singletons-appdelegates-and-top-level.html

[5] https://github.com/akosmasoftware/Senbei/tree/master/Classes/Helpers/SBNetworkManager

[6] https://github.com/akosmasoftware/Senbei/blob/master/Classes/Helpers/SBNotifications.h

One Subclass per Request

I create subclasses of ASIHTTPRequest for each kind of network request that my app requires (typically, for each backend REST interaction or SOAP endpoint).

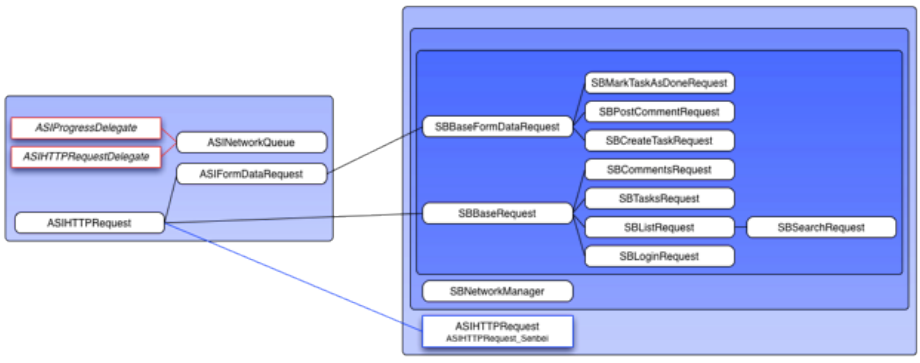
I also create a base class for all the requests of the application; this allows to centralize some shared behavior in the base class, which proves handy while extending and refactoring your network code.

In our example, Senbei has a base class for all the GET requests in the application, called SBBaseRequest[7]. There is another base request, called SBBaseFormDataRequest[8], which is used for requests that use the POST and PUT verbs (used to create and modify resources on the server).

There is also a category on ASIHTTPRequest, to add some methods to any request create on the system; this is required because SBBaseRequest inherits from ASIHTTPRequest, while SBFormDataRequest inherits from ASIHTTPFormDataRequest, which also inherits from ASIHTTPRequest. The category on the latter allows to inject some common behavior in a way that classic inheritance does not allow per se.

For every network interaction in the server, there is a dedicated class available for the SBNetworkManager; the code is easy to understand, and the responsibilities are separated and well defined. Should the system be extended in the future, the extension mechanism will naturally fit any new request, in a horizontal fashion.

The following class diagram (generated from the Xcode project using the excellent OmniGraffle[9] application) shows how the system is structured (you can click the diagram to see a larger version).



[10]

[7] https://github.com/akosmasoftware/Senbei/blob/master/Classes/Helpers/SBNetworkManager/Requests/SBBaseRequest.h

[8] https://github.com/akosmasoftware/Senbei/blob/master/Classes/Helpers/SBNetworkManager/Requests/SBBaseFormDataRequest.h

[9] http://www.omnigroup.com/products/omnigraffle/

[10] diagram-large.png

Polymorphism to the Rescue

The network manager doesn't know what to do with the result of each request; hence, it just calls a method **on the request**. Remember, requests are subclasses of ASIHTTPRequest, so you can just put the code that manages the result of the request (typically, deserialization of JSON or XML into real objects, triggering other network connections, updating Core Data stores, etc). Putting the code into each separate request subclass, using a polymorphic method with a common name accross request classes, makes it very easy to debug and manage them.

In our example, the SBNetworkManager calls the "processResponse" method in each subclass. This method has an empty implementation in our category for ASIHTTPRequest, and each individual subclass performs a different set of operations; some will parse XML, some will just post a notification; the separation of the logic in each subclass makes it easy to debug, document, maintain and extend the system.

Usage

Every time one of my controllers requires some data (refresh, viewDidAppear, etc), the network manager creates an instance of the required ASIHTTPRequest subclass, and then adds it to the queue.

Whenever a request finishes or fails, the network manager is called (remember, the network manager is the queue's delegate). In turn, the network manager calls a method on the request itself, delegating the task of the processing of the response to each subclass.

In Senbei, every method of the SBNetworkManager class just creates an instance of a dedicated SBBaseRequest subclass, and pushes it into the wrapped network queue.

Notifications

The network manager notifies the controllers above about interesting events using notifications; using a delegate protocol is not a good idea, because in your app you typically have many controllers talking to your network manager, and notifications are more flexible.

However, as with always with notifications, using them requires planning, naming conventions and documentation. Code using notifications might be complex to maintain, because the dependencies are not obvious at first hand; that's the price of their flexibility. In Senbei, notifications are all defined in the same file, so that different components can use the same constants throughout the application. The names of the notifications are clear and express the purpose and circumstance of each one.

Since iOS 4 there is also the possibility of using blocks as callback notifications, but then again, I think they just offer an alternative to

delegate protocols; notifications are much more flexible, as many different objects can be notified of the same event (and receive the same information through userInfo dictionaries) at once.

Conclusion

This is how I've been writing many network-bound apps for the past few years, and frankly it has worked pretty well so far. I can extend the system horizontally, adding more ASIHTTPRequest subclasses as I need them, and the core of the network manager stays intact. The responsibilities is clearly separated, and the class and notification names give a pretty good idea of the purpose of each request.

One problem that I haven't yet solved with this architecture (and one that a commenter of my StackOverflow answer points out) is finding a way to test the system; probably using mock objects, we could simulate different network conditions, and integrate this knowledge with automated tests.

I hope that this architecture is useful to you too! I look forward to read your comments below.

RooiFonts 1.1 in the App Store!

Adrian Kosmaczewski

2011-09-22

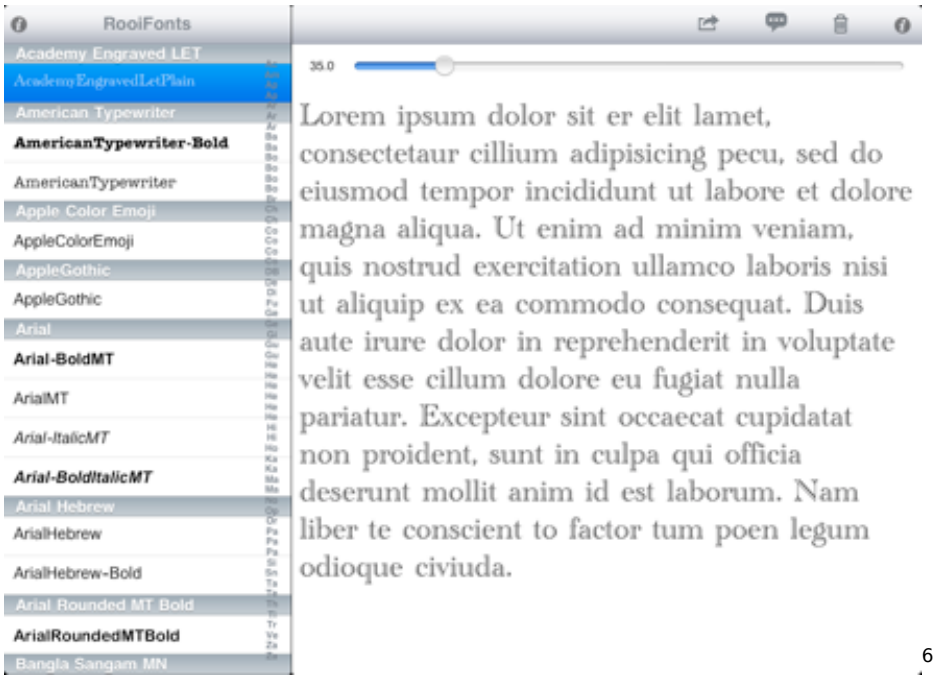
The update frenzy continues! This is a long due update, one that we have been pushing back for a while; finally, RooiFonts 1.1[1] is ready to be downloaded from the App Store! RooiFonts our popular font viewer that allows users to browse, compare, share and learn about the fonts installed in iOS.

This new version is now a Universal application, that runs both on the iPad, the iPhone and the iPod touch! It also features nice new icons, compatible with the Retina display of the iPhone 4, and we've also updated the application for compatibility with iOS 4.0+ only[2].

As usual, this is a free update for existing customers. Get it now![3] We are also going to give away some promo codes on the RooiFonts Twitter account[4], so stay tuned and follow us!

[5]

[1] http://rooifonts.com/
[2] /blog/dropping-support-for-iphone-os-3-x/
[3] http://itunes.com/apps/rooifonts
[4] https://twitter.com/RooiFonts
[5] http://itunes.com/apps/rooifonts



[6]

[6] http://itunes.com/apps/rooifonts

Notitas 2.0: A Major Milestone!

Adrian Kosmaczewski

2011-09-26

Notitas 2.0[1] has been approved during the weekend, and is available in the App Store!

[2]

This is a major revision of our review-winning application[3], including a much awaited adaptation to the iPad! That's right, you can now create, read and edit your notes directly on the iPad. This new application is as simple to use as the iPhone version, but with the added benefit of allowing you to move notes freely around the interface, ordering and changing them as you need. A full-sized map shows the location of all your notes, helping you keep track of your ideas and their location.

The iPad version come bundled in the same package as the iPhone application, so that current owners will get the iPad version for free, the next time you update. We have also updated the graphics of the iPhone version to support the newest Retina displays, and finally, we have updated the application to support only iOS 4.0 and higher, as explained previously in this blog[4].

[1] http://muchasnotitas.com/Notitas/

[2] http://itunes.com/apps/notitas

[3] http://www.lexpress.fr/actualite/high-tech/notitas-im-plus-old-booth-premium_791232.html

[4] /blog/dropping-support-for-iphone-os-3-x/

[5]

Get your copy of Notitas now![6] We would love to hear your feedback! We are going to continue to push new features into it, so stay tuned to our Twitter account @MuchasNotitas[7]!

[5] http://itunes.com/apps/notitas
[6] http://itunes.com/apps/notitas
[7] http://twitter.com/MuchasNotitas

Auto Portrait

Adrian Kosmaczewski

2011-09-26

I’m looking for the face I had, before the world was made

W. B. Yeats.

Made with Artrage for iPad.

Senbei 1.3 hits the App Store!

Adrian Kosmaczewski

2011-09-27

Our application Senbei[1] has been updated to version 1.3! Senbei is the iPhone client of the popular Fat Free CRM[2] application by Michael Dvorkin[3], written in Ruby on Rails[4].

5

In this version we have updated the code for compatibility with iOS 4.0 and later, and also with graphics that take advantage of the latest Retina displays. We have also solved a number of small but annoying bugs that were scattered throughout the application.

Get Senbei from the App Store[6] or check out the source code in Github![7] Senbei features an implementation of the proposed network request architecture[8] we've talked about in this blog before, so be sure to check that part of the code as well.

[1] http://itunes.com/apps/senbei
[2] http://www.fatfreecrm.com/
[3] http://twitter.com/mid
[4] http://rubyonrails.org/
[5] http://itunes.com/apps/senbei
[6] http://itunes.com/apps/senbei
[7] https://github.com/akosma/Senbei
[8] /blog/a-proposed-architecture-for-network-bound-ios-apps/

Announcing the swissinfo iPhone Application!

Adrian Kosmaczewski

2011-09-28

We are thrilled to announce the general availability of version 2.0 of the swissinfo application[1], this time bundled as a Universal Application, including an iPhone version together with the iPad version!

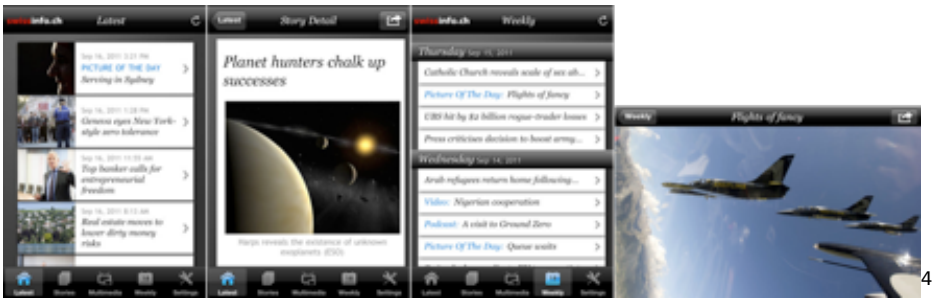
[2]

This app includes articles, high quality videos, audio slideshows, podcasts and galleries from the Swiss Broadcasting Corporation. Designed for maximum comfort, the app also includes a weekly review to keep in touch with the essential Swiss news.

This Swiss Broadcasting Corporation application is available in 9 languages: English, Chinese, Arabic, Japanese, Spanish, Portuguese, French, German and Italian.

Check it out on the App Store![3]

[1] http://itunes.apple.com/us/app/swissinfo.ch/id407776916?mt=8
[2] http://itunes.apple.com/us/app/swissinfo.ch/id407776916?mt=8
[3] http://itunes.apple.com/us/app/swissinfo.ch/id407776916?mt=8



[4]

PS: The application is launched simultaneously today with the new Android application[5], featuring a very similar look and feel as the iPhone version. Check it out as well!

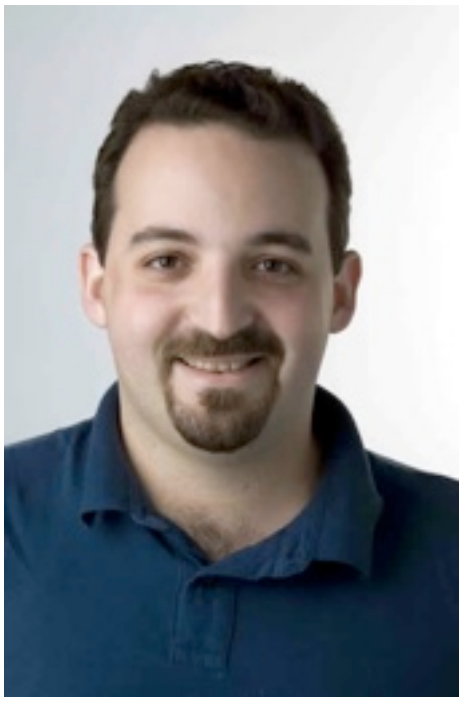
[4] http://itunes.apple.com/us/app/swissinfo.ch/id407776916?mt=8
[5] https://market.android.com/details?id=ch.swissinfo.mobilelite&hl=en

“jQuery Mobile: multiplatform mobile webapps” by Maximiliano Firtman!

Adrian Kosmaczewski

2011-09-28

Invisible GmbH[1] and akosma software[2] are thrilled and excited to present the talk “jQuery Mobile: multiplatform mobile webapps” by Maximiliano Firtman[3] at Hub Zürich[4] on Friday, October 21st at 18h, with the sponsorship of /ch/open[5].



Mobile development is here. Everyone needs apps and mobile websites.

[1] http://invisible.ch/

[2] http://akosma.com/

[3] http://techup.ch/370/jquery-mobile-multiplatform-mobile-webapps-by-maximiliano-firtman

[4] http://www.hubzurich.org/

[5] http://www.ch-open.ch/

Here comes jQuery Mobile[6], an easy to use, standard and open sourced framework that allow us to create mobile apps in minutes. Compatible with iOS, Android, webOS, Symbian, Windows Phone BlackBerry and optimized for smartphones and tablets, this framework is ideal for any web designer or developer who wants to create mobile experiences.

With this framework, we can create browser-based experiences and webapps and also native apps compiled using hybrids techniques. Therefore, it's perfectly compatible with store publishing, such as Apple AppStore, Android Market, HP AppCatalog, RIM AppWorld and Nokia's Ovi Store.

In this session, we will cover:

- The mobile ecosystem today
- What is the problem with standards and platforms
- HTML5 Mobile overview
- jQuery Mobile and why you don't need to know jQuery to use it.
- Creating our webapps using HTML5 standard code
- Relation between jQuery Mobile and native webapps (hybrids) such as PhoneGap
- Samples

Max Firtman is a mobile+web developer, trainer, speaker and writer. He is Nokia Developer Champion, Adobe Community Champion and founder of ITMaster Professional Training. He wrote many books, including "Programming the Mobile Web"[7] published by O'Reilly Media and the upcoming book "jQuery Mobile: Up and Running"[8]. He has a blog about mobile web development at <http:></http:> and he maintains the website <http:></http:>. He is a frequent speaker at conferences, including OSCON, Breaking Development Velocity Conference, Google Developer Day, Nokia Developer Days, Campus Party Europe and many other events around the world. He has received different recognitions, including Nokia Developer Champion yearly since 2006; Adobe Community Champion in 2011, and a Google recognition for being one of the most innovative mobile developers.

View the information about the event in techup.ch[9] and the Cominmag agenda[10].

[6] http://jquerymobile.com/

[7] http://shop.oreilly.com/product/9780596807795.do

[8] http://shop.oreilly.com/product/0636920014607.do

[9] http://techup.ch/370/jquery-mobile-multiplatform-mobile-webapps-by-maximiliano-firtman

[10] http://www.cominmag.ch/agenda/event/techup-ch-jquery-mobile-multiplatform-mobile-webapps-par-maximiliano-firtman/

Mobile Application Training in South Africa

Adrian Kosmaczewski

2011-09-30

The next two weeks I will be giving an extensive set of mobile application trainings in South Africa, organized jointly by immedia and akosma software!

Have you always wanted to learn how to create a mobile app? Looked at Evernote and thought, “Hey, I can do that!” We have embarked on an exciting training initiative aimed at providing South African businesses and individuals with the skills they need to become global app creators.

Courses will be held in Johannesburg, Pretoria and Durban, and will be provided[1] by Anice Hassim, head strategist of immedia; Kishyr Ramdial, specialist cloud and mobile app developer; and myself.

There are courses targeted at app developers, dev beginners, and executives – to ensure that you learn exactly what you need to from our programme.

Check out the complete program here: <http:></http:>; there are still some places left, hurry up!

[1] http://www.immedia.co.za/courses/trainers.html

Mobile Development with jQuery, Sencha and PhoneGap, 15-17 November 2011

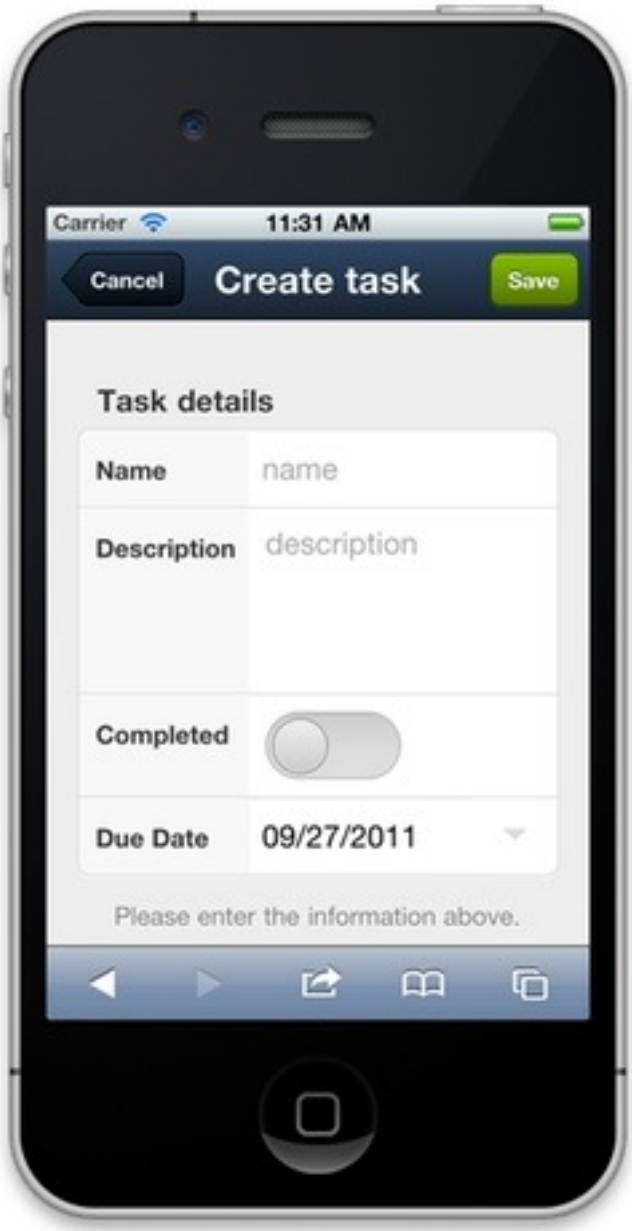
Adrian Kosmaczewski

2011-10-03

Invisible GmbH and akosma software are thrilled to announce their latest collaboration: a three day training about mobile web app development using jQuery, Sencha Touch and PhoneGap![1]

Today, having a mobile application online is a must. But there are multiple platforms to write for, each with their own language, idioms and pitfalls. Luckily there is a simple solution that allows to write once and deploy on all modern mobile devices: HTML5 and JavaScript.

[1] http://mobile-training.ch/



Easy, quick dev of “native” applications

This three day intensive course takes you from being a web developer to being a mobile developer. We take you through the basics of writing HTML5 applications for mobile devices, cover the additional APIs that allow you to access the functions of the devices (like storage, geo-location, accelerometers) and put you in control of deploying an application to either iOS or Android devices.

Who

Jens-Christian Fischer, InVisible GmbH

Jens-Christian started writing software in the late 1980s and been working on Web applications since the mid 1990s. The last 6 years he has been developing, writing and teaching Ruby on Rails and other web related technologies. Jens-Christian is the CEO of InVisible GmbH, a Zurich based web development agency.

Adrian Kosmaczewski, akosma software

Adrian has been writing software for the past 20 years. He started working professionally in 1996, riding the first and second waves of the web. He started writing Cocoa applications for the Mac in 2002, and has been writing iOS apps since he returned from WWDC 2008. Adrian is the founder of akosma software, with a strong focus in all things iOS.

Course Details

Location

The 3 day course will be held the 15. – 17. November 2011, in Zurich.

Price

- Regular Price: CHF 1650.–
- Early Bird Price: CHF 1400.– (a 15% discount) if ordered before 15.10.2011.
- Multiple people from the same company? Get a 10% discount for the second person.
- Members of /ch/open receive a 15% discount (not cumulative with early bird price)

Included in price

- 3 days of intensive hands-on training in a small group, with plenty of time to talk to the instructors
- Comprehensive Documentation
- Full lunch meal
- “There has to be food” – plenty of snacks and drinks during the day

More Information

Don’t hesitate to check <http:></http:> for updates, or to contact us[2] for more information.

[2] /about/

Article in Cominmag

Adrian Kosmaczewski

2011-10-17

The October issue of Cominmag[1] includes an article in French by akosma software about the new shape of the mobile market in Switzerland, check it out! And you can read it online too![2]

[1] http://www.cominmag.ch/
[2] http://www.cominmag.ch/web-mobile/

Attending SenchaCon 2011

Adrian Kosmaczewski

2011-10-19

We are very happy to announce that next week we will be attending SenchaCon 2011[1], the most important event of the year around the Sencha[2] frameworks. This will happen in Austin, Texas, from the 23rd to the 26th of October.

[3]

We are keen Sencha developers and we think this platform has a huge potential. With the recent announcements of Sencha Touch Charts[4] and a possible new "Touch" version of the incredible Ext Designer[5], this platform will take web apps to new limits, particularly in the enterprise world.

We will be sharing information about the new features of Ext.js 4[6] and Sencha Touch[7] through this website, our Twitter account[8] and our Facebook page[9].

Be sure to follow & "like" us, and stay tuned!

[1] http://www.sencha.com/company/events/senchacon-2011/
[2] http://www.sencha.com/
[3] http://www.sencha.com/
[4] http://www.sencha.com/products/touch/charts/
[5] http://www.sencha.com/products/designer/
[6] http://www.sencha.com/products/extjs/
[7] http://www.sencha.com/products/touch/
[8] http://twitter.com/akosmasoftware/
[9] https://www.facebook.com/akosmasoftware

Guia Irresponsable Y Atorrante Para Conocer Nueva York

Adrian Kosmaczewski

2011-10-29

El otro día escribí esto para una amiga que está por viajar a Nueva York, y me gustó, así que aquí va.

Nueva York es la ciudad por antonomasia. Yo estuve ahí en el 2000 y 2001, y después en el 2010 con Clau para ir a comprar iPads :)

Los lugares turísticos clásicos, no te los podes perder; el Empire State, Central Park, la estatua de la libertad (me dijeron que es mucho mas petisa de lo que uno cree, yo nunca fui ahí), Times Square.

Consejos

Si llegan por el aeropuerto de Newark (New Jersey, cruzando el río Hudson), hay un tren que los deja en el Penn Station, en pleno centro.

Si llegan por el JFK, creo que hay un subte que te lleva a Manhattan. Pero no recuerdo, creo que me tome un taxi.

Es una ciudad hecha para caminar, a pesar del tamaño gigantesco que tiene. Pero fuera de broma, llevense buen calzado y a patear. Tiene una dimensión humana que es alucinante. Muy buena onda. Es, además, relativamente segura.

Preparen la cámara de fotos. Llevense batería de repuesto cargada y una tarjeta de memoria extra. La van a necesitar.

Paseos que hice que me encantaron

Cuando subí a las Torres gemelas, esto fue en junio del 2000, después me fui caminando hasta el puente de Brooklyn, que tiene un sector para peatones completamente aislado de los autos, y es un paseo genial. Eso si, agarrate porque el puente mide varias centenas de metros, creo yo casi un km de largo (cuando veas el cartel que dice que fue inaugurado en 1863, si no se te caen las medias, me voy a enojar). Es muy largo. Pero del otro lado es precioso, hay un paseo muy bonito del que se ve Manhattan y esta muy lindo, no hay que perderselo.

Después me fui caminando del puente de Brooklyn hasta Chinatown, que es súper pintoresco. Es una zona de Manhattan que no tiene edificios de no mas de 10 pisos de alto (la geología de la isla no permite rascacielos sino en los extremos sur y centro-norte de la isla) y de ahí podes ir a Little Italy, cruzar Washington Square y caminar por la quinta avenida. No te digo de seguir caminando por ahí hasta Times Square porque es un trecho largo.

La arquitectura de esa ciudad es algo fuera de lo común. Te va a dar torticolis de tanto mirar para arriba. Mi edificio preferido es el Chrysler, con su cúpula plateada art deco de los años 30. Precioso.

Si van a subir al Empire State, tomense toda la mañana para ello. Las colas son largas, y subir y bajar les va a tomar al menos 3 horas, paciencia!

Tomense el subte: fuera de broma, las estaciones miden como 2 cuadras y no podes creer cuantas líneas hay. Hay pases diarios para entrar y salir cuantas veces quieran.

Vayan a Times Square de noche. No de día; de noche. Fuera de broma, haceme caso. Cuando estén ahí, acordate de la versión del tema de Alicia Keys “Empire State of the Mind” con Jay Z. Cantenlo. Jajaja fuera de broma, las fichas caen en ese instante.

Central Park se merece un día entero, es enorme y da para un lindo picnic. Yo conozco solamente la punta sur. Hay muchísimas callecitas, lagos, es precioso.

Justamente, pegado a la punta sur del Central Park, esta el Apple Store de la quinta avenida :) aunque no compren nada, metete nomas para ver. Es alucinante.

Por esa zona también, dos o tres cuadras mas al sur, esta el Rockefeller Center. No se lo pierdan.

Tomense un taxi, preferentemente en hora pico. Tomenlo en Washington Square y diga le de ir hasta la 70th o la 80th, así ven toda la quinta avenida. Después vuelven en subte si quieren.

Metanse en el bar del Hilton o del W hotel (a unas cuadras del Empire State) y tomense un martini seco. Como en las películas, con los vasos triangulares. Con jazz de fondo y todo. Súper romántico.

Vayan al teatro en Broadway. Es algo que tengo muchas ganas de hacer algún día.

En la calle, comanse un buen hot dog. Insisto. Tipo 2 de la tarde, en la esquina de Madison y la 57ava. O por ahí. Preferentemente al lado de una boca de tormenta humeante o de unos canas de la NYPD. Con un poco de suerte en ese momento pasan los bomberos del NYFD y el hombre araña haciendo piruetas.

Si tienen tiempo, vale la pena que se alquilen un auto y vayan para el norte del estado de NY, hacia Albany. Saliendo de la ciudad, media hora después estas en un paisaje de bosques y ríos que es precioso.

Es la entrada de los montes Apalaches, una preciosura. Yo anduve por ahí alla por el 2000.

Los yanquis se van de vacaciones a NYC como si fuese ir a París. Es muy cómico ver los buses de gente que acaba de llegar de Oklahoma o Arkansas para visitar la ciudad, todos gorditos, con gorra de béisbol, y haciendo fila en la entrada de cuanto lugar he enumerado más arriba.

NY es la ciudad mas grande de Irlanda. Fuera de broma, la herencia irlandesa es muy fuerte y hay pubs por todos lados. Tomense una buena Guinness después de un día de caminata y brinden agradeciendome toda esta data :)

Que la pasen genial! Es una ciudad fantástica.

News for this week: Training + Interview

Adrian Kosmaczewski

2011-11-14

This week we have exciting news to announce!

The mobile web training in Zürich[1] we prepared with Invisible GmbH[2] / Simplificator GmbH[3] has SOLD OUT! How cool is that? We are going to spend three days talking about Sencha Touch[4], PhoneGap[5] and jQuery Mobile[6]. We are going to announce more dates in the future, and also new locations for 2012! We believe that the mobile web is growing and we want to help you be there as fast as possible. Stay tuned!

And the second announcement is that tonight Adrian will appear in an interview, shown as part of TTC "Toutes Taxes Comprises"[7], a program of the French-speaking Swiss Television channel TSR 1[8]. Check it out tonight at 8:10pm!

[1] http://mobile-training.ch/
[2] http://invisible.ch/
[3] http://simplificator.com/
[4] http://www.sencha.com/products/touch/
[5] http://phonegap.com/
[6] http://jquerymobile.com/
[7] http://www.tsr.ch/emissions/ttc/3481214-a-fond-les-applis.html
[8] http://www.tsr.ch/

Interview on the Swiss TV

Adrian Kosmaczewski

2011-11-27

In the context of the French-speaking Swiss television program “TTC” (“All Taxes Included”) Adrian has been interviewed about mobile applications and the economics around it.

You can check the complete interview online![1] (and, as usual[2], Adrian’s family name has suffered a bit in the process :)

[1] http://www.tsr.ch/emissions/ttc/3481214-a-fond-les-applis.html
[2] /blog/interview-in-the-bolivian-tv/

Destrozando El Apellido

Adrian Kosmaczewski

2011-11-27

Aquí inicio la sección oficial de mis apariciones en la TV con apellidos destrozados!

Bolivia:

Suiza:

Decí que existe el copy-paste, que si no...

QCon London 2012: Call for Papers in the Cross-Platform Mobile Track

Adrian Kosmaczewski

2011-12-12

Just like last year[1], I am thrilled to announce that I will be the host of the Cross Platform Mobile track[2] in QCon London 2012[3]!

QCon is a series of conferences taking place in San Francisco[4] and London every year. It is a joint venture between Trifork[5] and InfoQ[6].

We are currently working towards the 2012 edition of the London event, gathering input from you, to get the best speakers and the best content. We will be focusing on web technologies and frameworks, as well as standards and also design. If you want to participate as a speaker, or have someone you would like to recommend, just use the contact form in this site[7]! I look forward to hearing from you.

[1] /blog/qcon-london-2011-call-for-papers-in-the-mobile-track/
[2] http://qconlondon.com/london-2012/tracks/show_track.jsp?trackOID=569
[3] http://qconlondon.com/
[4] http://qconsf.com/
[5] http://www.trifork.com/
[6] http://www.infoq.com/
[7] /about/

Student application on the App Store: TroisXRien

Adrian Kosmaczewski

2011-12-13

Remember Sudokulus? It's the application created by one of my students[1] in a previous training. Well, today I have the pride and the joy to announce the second application created by one of our students, and published on the App Store!

[2]

The application is called TroisXRien[3], and it is an application that kids can use to learn arithmetics; select one or many exercises, and kids repeat them until they don't make any more mistakes! Parents can follow the evolution of their kids closely, as the application can send them an email when the training is done.

A very nice idea, beautifully illustrated – Vincent is not only a great school teacher, but also a tremendous cartoonist! (by the way, he's the author of these drawings about akosma[4] that you can check in our Facebook page[5] :))

[1] /blog/student-application-on-the-app-store-sudokulus/

[2] http://itunes.apple.com/ch/app/troisxrien/id465719143?l=fr&mt=8

[3] http://itunes.apple.com/ch/app/troisxrien/id465719143?l=fr&mt=8

[4] https://www.facebook.com/media/set/?set=a.10150426423201100.383059.18404 6196099&type=1

[5] https://www.facebook.com/akosmasoftware

Congratulations Vincent! Let's give his app some download love! It already features lots of 5 star ratings, let's give him more of them!

[6]

[6]http://itunes.apple.com/ch/app/troisxrien/id465719143?l=fr&mt=8

Swiss App Awards 2012: Submit your app now!

Adrian Kosmaczewski

2011-12-19

I'm honored to announce that I will be part of the jury of the next Swiss App Awards 2012[1], to be held in Zürich, on January 27th, 2012!

The event is a gathering of the many designers, developers and companies that contribute to excellence, innovation and achivement in the Swiss app development landscape. The Awards show is looking to celebrate the most outstanding apps developed or updated in 2011 and owned by Swiss companies. The apps that will be rewarded at the Swiss App Awards 2012 are not only innovative but do also demonstrate excellence both on technical and design related features.

I will be sharing the duty of jury with:

- Janne Jul Jensen, PhD and expert in UX design;
- Katja Neumann, usability engineer at Zühlke;
- Jørn Larsen, co-founder and CEO of Trifork[2]; and
- Martin Coul, founder of the Swiss Mobile Basecamp[3].

You can submit your application[4] (iOS, Android, Windows Mobile, web, any kind and in any language!) or just register as an attendee[5] on the website of the event[6]. You can also follow the official Twitter account[7] of the Swiss App Awards for more information.

We look forward to see your applications!

[8]

[1] http://swissappawards.ch/
[2] http://www.trifork.com/
[3] http://www.mobicamp.ch/
[4] http://form.jotform.com/form/13471218230
[5] https://secure.trifork.com/swiss-app-awards-2012/registration/
[6] http://swissappawards.ch/
[7] http://twitter.com/SwissAppAwards
[8] http://swissappawards.ch/

Announcing EERV cal, an iPhone, Android and Web Application!

Adrian Kosmaczewski

2011-12-20

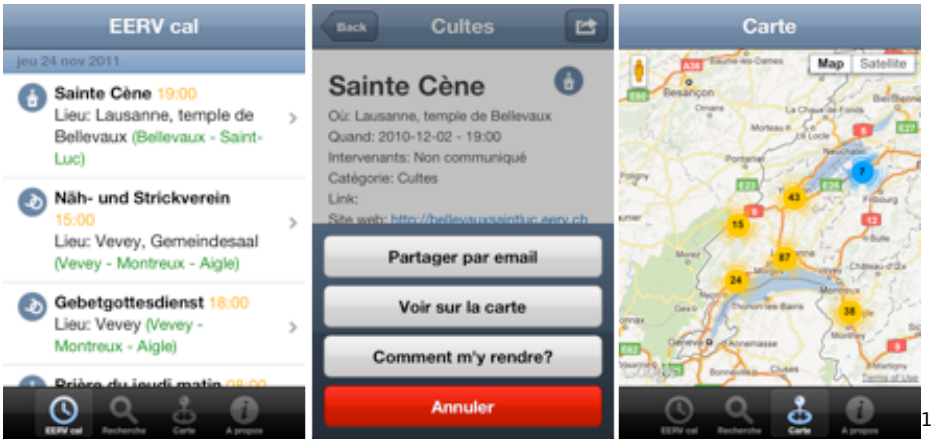
We are ecstatic to announce the general availability of the EERV cal[1] mobile application, created with Sencha Touch[2] and PhoneGap[3], and available as an HTML5 web application[4], and also on the App Store[5] and the Android Market[6]!

[7]

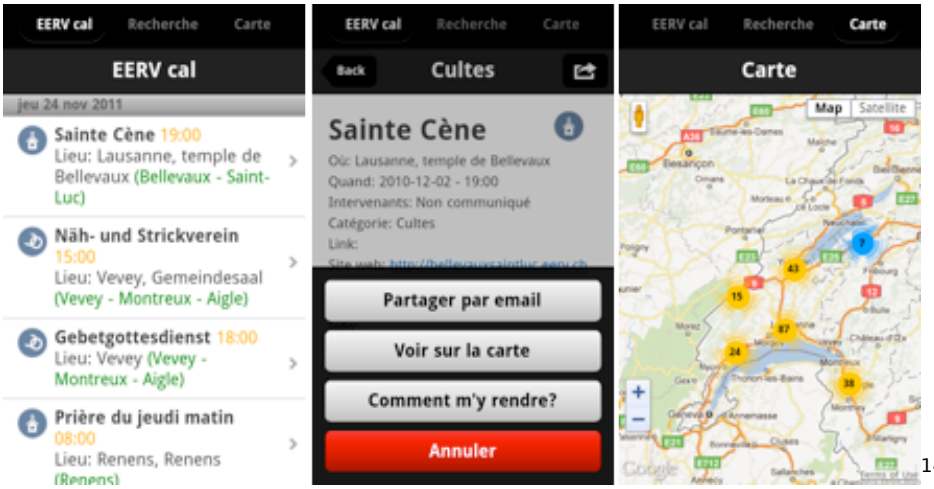
This app has been built for the local protestant church, the Eglise évangélique réformée du canton de Vaud[8] (EERV), and features the complete agenda[9] of events proposed throughout the region.

Check it out on the App Store[10] the Android Market[11] or install the web app directly from your browser[12]!

[1] http://eerv.ch/
[2] http://www.sencha.com/products/touch
[3] http://phonegap.com/
[4] http://eerv.ch/mobile/
[5] http://itunes.apple.com/ch/app/eerv-cal/id483342559?l=en&mt=8
[6] https://market.android.com/details?id=ch.eerv
[7] http://eerv.ch/
[8] http://eerv.ch/
[9] http://eerv.ch/agenda
[10] http://itunes.apple.com/ch/app/eerv-cal/id483342559?l=en&mt=8
[11] https://market.android.com/details?id=ch.eerv
[12] http://eerv.ch/mobile/



[13]



[14]

[13] http://itunes.apple.com/ch/app/eerv-cal/id483342559?l=en&mt=8
[14] https://market.android.com/details?id=ch.eerv

Makeover

Adrian Kosmaczewski

2011-12-22

I thought it was time to change the look and feel of this blog. Well, it's done. And I've gone as minimalistic as possible.

By the way, no more comments. At all. Ever again. I've had enough of spam, hatred, flame wars, and idiots of any kind. For those who played the game, thanks for reading and commenting in a constructive way.

A happy life to all of you.

More Mobile Application Training in South Africa

Adrian Kosmaczewski

2011-12-22

Just like last October[1] I'll be back in South Africa next February, for a whole series of mobile application trainings organized jointly by immedia[2] and akosma software!

Have you ever wanted to learn how to create a mobile app? Looked at Evernote and thought, "Hey, I can do that!" We have embarked on an exciting training initiative aimed at providing South African businesses and individuals with the skills they need to become global app creators.

Courses will be held in Cape Town, Johannesburg and Durban, and will be provided[3] by Anice Hassim, head strategist of immedia; Kishyr Ramdial, specialist cloud and mobile app developer; and myself.

There are courses targeted at app developers, dev beginners, and executives – to ensure that you learn exactly what you need to from our programme.

Check out the complete program here: <http:></http:> and sign up sending an email to ; hurry up!

[1] /blog/mobile-application-training-in-south-africa/
[2] http://www.immedia.co.za/
[3] http://www.immedia.co.za/courses/trainers.html

Retrospective 2011

Adrian Kosmaczewski

2011-12-23

2011 was another terrific year for akosma software, a year of growth, surprise, learning, teaching, and travel. We met fantastic new people on the road and we would like to thank all of our friends, our customers, the readers of this blog, our followers on Twitter[1], Facebook[2], Github[3], LinkedIn[4] and countless other networks for their support and feedback throughout the year! May all of you have a healthy and happy 2012.

And now let us share with you some highlights of 2012!

Training

akosma software has established in 2011 a solid and successful training business, in the domains of iOS and mobile web applications:

[5]

- We started the year with a successful iOS Advanced Training[6] in February.
- Then we continued with a long training, from February to June, to teach iOS to school teachers in the Canton of Vaud, in Switzerland.
- Later this year we went to teach iOS development in South Africa[7] during October, in a series of trainings co-organized with immedia[8].

[1] http://twitter.com/akosmasoftware
[2] https://www.facebook.com/akosmasoftware
[3] https://github.com/akosmasoftware
[4] http://www.linkedin.com/company/akosma-software
[5] http://www.immedia.co.za/
[6] /blog/advanced-ios-4-2-training-course-zurich-february-7th-and-8th-2011-enroll-now/
[7] /blog/mobile-application-training-in-south-africa/
[8] http://www.immedia.co.za/

- We organized with Simplificator GmbH the first Swiss training on jQuery Mobile, Sencha Touch and PhoneGap of the country[9], which was also a huge success in November, in Zürich.
- Last but not least, we coached our friend Mathieu Tendon to get his degree thesis, implementing an Android version of Senbei!

[10]

The result of these efforts were an incredible suite of apps created by some of our students, and now available on the App Store:

- Sudokulus[11] by Pieter Muller, an incredible application that allows us to quickly solve sudoku puzzles!
- TroisXRien[12] by Vincent Zeller, a very nice app to teach maths to children!

Conferences

2011 was, without any doubt, the year of the conference; not only have we participated in lots of them as speakers, we could also be the host of an incredible mobile track and even co-organized one in Zürich!

[9] /blog/mobile-development-with-jquery-sencha-and-phonegap-15-17-november-2011/
[10] /blog/student-application-on-the-app-store-troisxrien/
[11] /blog/student-application-on-the-app-store-sudokulus/
[12] /blog/student-application-on-the-app-store-troisxrien/

13

- It all started in January, when we participated at the international OOP conference[14] in Münich.
- In March Adrian was the host of the mobile track[15] in the prestigious QCon Conference London! It was an incredible day, and you can check the videos of the sessions in the mobile track directly on the InfoQ site: Mike Lee[16]'s, Graham Lee[17]'s and his own presentation[18] are there.
- In April we went to Göteborg, Sweden to talk in the Scandinavian Developer Conference 2011[19]!
- In June we were invited by the CAINCO in Santa Cruz[20], Bolivia, to talk about mobile applications!
- In October, together with Jens-Christian Fischer[21] from Simplificator[22] we brought the biggest expert of the world of the mobile web, Maximiliano Firtman, to talk in Zürich[23]!
- Later that same month we attended SenchaCon in Austin, Texas[24], to learn more about the incredible Sencha Touch framework.
- Finally, in November we went to the Mobile Developer Summit in Bangalore, to give several different talks[25] about iOS and the

[13] http://www.sigs-datacom.de/oop2011/konferenz/sessiondetails.html?tx_mwconferences_pi1%5BshowUid%5D=382&tx_mwconferences_pi1%5Bpointer%5D=0&tx_mwconferences_pi1%5Bmode%5D=1&tx_mwconferences_pi1%5Bs%5D=0

[14] http://www.sigs-datacom.de/oop2011/konferenz/sessiondetails.html?tx_mwconferences_pi1%5BshowUid%5D=382&tx_mwconferences_pi1%5Bpointer%5D=0&tx_mwconferences_pi1%5Bmode%5D=1&tx_mwconferences_pi1%5Bs%5D=0

[15] http://qconlondon.com/london-2011/tracks/show_track.jsp?trackOID=417

[16] http://www.infoq.com/presentations/Making-Apps-That-Dont-Suck

[17] http://www.infoq.com/presentations/Mobile-App-Privacy

[18] http://www.infoq.com/presentations/Introduction-to-iOS-Software-Development

[19] http://www.scandevconf.se/2011/conference/speakers/arian-kosmaczewski/

[20] /blog/foro-de-tecnologia-en-santa-cruz-de-la-sierra-bolivia-el-16-de-junio/

[21] http://twitter.com/jcfischer

[22] http://simplificator.com/

[23] /blog/jquery-mobile-multiplatform-mobile-webapps-by-maximiliano-firtman/

[24] /blog/attending-senchacon-2011/

[25] /blog/speaking-at-the-mobile-developer-summit-in-bangalore/

mobile web.

Consulting and Development

This year we strengthtened our collaboration with clients, providing development and consulting services to industry leaders in Switzerland and abroad:

[26]

- We started a long and fruitful collaboration with immedia[27], a leading provider of mobile application[28] development services in South Africa, by providing them with strategy and development consulting services.
- We extended the swissinfo iPad application into a universal iOS application, featuring an iPhone application[29] too!
- The International Federation of Red Cross and Red Crescent Societies[30] asked us for help in the definition of the mobile technologies for a humanitarian project, which consisted in an extensive report.
- We helped the local protestant church, the Eglise évangélique réformée du Canton de Vaud, to create their brand new mobile application[31], built using Sencha Touch[32] and PhoneGap[33].
- We worked with many other customers who have explicitly asked us to remain anonymous, but we thank them for their trust and their business! We are proud and happy to have worked with them.

[26] /blog/announcing-eerv-cal-an-iphone-android-and-web-application/
[27] http://www.immedia.co.za/
[28] /blog/immedia-the-leader-mobile-solutions-provider-in-south-africa/
[29] /blog/announcing-the-swissinfo-iphone-application/
[30] http://www.ifrc.org/
[31] /blog/announcing-eerv-cal-an-iphone-android-and-web-application/
[32] http://www.sencha.com/products/touch
[33] http://phonegap.com/

Products

Our apps also received some update love in 2011! We provided new versions of all of our iOS applications on the App Store, in many cases introducing the long-awaited iPad versions:

 [34]

- bluewoki[35] has jumped to version 2.0[36] and introduced support for wifi networks!
- Notitas[37] has finally become a universal application! The new iPad[38] version was a welcome addition and has had a very nice effect on sales so far.
- RooiFonts[39] has also got an iPad version![40]
- Senbei was updated[41] with many bug fixes.
- Finally, our flagship open source project, nib2objc[42], has been updated to support new features of iOS 4.

On the Press

This year we have increased our presence in the media, showcasing our ideas and our opinion on the world of technology:

[34] http://muchasnotitas.com/
[35] http://bluewoki.com/
[36] /blog/bluewoki-2-0-hits-the-app-store/
[37] http://muchasnotitas.com/
[38] /blog/notitas-2-0-a-major-milestone/
[39] http://rooifonts.com/
[40] /blog/rooifonts-1-1-in-the-app-store/
[41] /blog/senbei-1-3-hits-the-app-store/
[42] https://github.com/akosma/nib2objc

43

- Early this year, our famous nib2objc project was featured in the popular open source blog “The ChangeLog”[44]
- The day before our talk at CAINCO, we were interviewed live on the Bolivian TV[45]
- We were featured in an article about the local mobile application market on the SonntagsZeitung[46], the most important newspaper of Switzerland!
- Our talk on QCon London 2011 was featured on InfoQ[47].
- We’ve been interviewed on the Swiss TV[48] as well, during a special emission about smartphone applications.
- And last but not least, we now have a regular column about mobile applications[49] in the Cominmag magazine[50]!

Other news

But that’s not all! 2011 brought lots of surprises and announcements:

[43] /blog/article-in-the-sonntagszeitung-die-neuen-schweizer-macher/
[44] /blog/nib2objc-featured-on-the-changelog/
[45] /blog/interview-in-the-bolivian-tv/
[46] /blog/article-in-the-sonntagszeitung-die-neuen-schweizer-macher/
[47] /blog/introduction-to-ios-software-development-video-on-infoq/
[48] /blog/interview-on-the-swiss-tv/
[49] /blog/article-in-cominmag/
[50] http://www.cominmag.ch/

[51]

- Early this year we moved to new offices[52] in Oron-la-Ville.
- The incredible team of moser design[53] created for us an incredible new corporate image[54], including a very cool set of logos and stationery!
- We opened up a new Twitter account[55] and a new Facebook page[56] for the company!
- We detected a huge change in the market, where the demand of mobile web apps is growing[57] and opens up very exciting new opportunities!
- During the year we dropped support for iOS 3.x[58] in all of our products and also for the projects we create for our clients.
- Finally, we also stated our policy of refusing to offshore projects[59].

[51] /blog/new-visual-identity/
[52] /blog/new-address/
[53] http://moserdesign.ch/
[54] /blog/new-visual-identity/
[55] http://twitter.com/akosmasoftware
[56] https://www.facebook.com/akosmasoftware
[57] /blog/a-shift-in-the-market-towards-mobile-web-apps/
[58] /blog/dropping-support-for-iphone-os-3-x/
[59] /blog/why-do-not-we-outsource-projects-overseas/

And What About Next Year?

2012 is already filled up with new exciting projects coming up!

- In January we will be part of the Jury[60] of the Swiss App Awards[61].
- In February we will be working with immedia[62] providing new mobile development training sessions[63] in South Africa.
- We will be hosting the Cross-Platform Mobile track[64] of QCon London 2012.
- We will participate in GOTOcon Copenhagen 2012[65] in May.

But that's not all!

- Great new projects already in preparation, to be announced soon!
- More trainings in Switzerland, both for iOS and the mobile web.
- …and much, much more!

Stay tuned as usual, and be very happy in 2012!

[60] /blog/swiss-app-awards-2012-submit-your-app-now/
[61] http://swissappawards.ch/
[62] http://www.immedia.co.za/
[63] /blog/more-mobile-application-training-in-south-africa/
[64] /blog/qcon-london-2012-call-for-papers-in-the-cross-platform-mobile-track/
[65] http://gotocon.com/cph-2012/

Kamgan Ukudigaa
Bon Nadal i feliç any nou
God jul og godt nytt år!
Zorionak eta urte berri on
Glædelig jul og godt nytår!
नये साल की हार्दिक शुभकामनायें
והבוט הנשו חמש דלומ גח
God jul och gott nytt år!
Joyeux Noël et bonne année!
Feliz Navidad y próspero año nuevo!
Merry Christmas and Happy New Year!
Hyvää joulua ja onnellista uutta vuotta!
즐거운 성탄절 보내세요 및 새해 복 많이 받으세요
Frohe Weihnachten und ein gutes neues Jahr
Честита Коледа! Щастлива Нова Година!
Masaplalang Bayung Banwa keko ngan!
Wesołych świąt i szczęśliwego nowego roku!
Chúc Giáng Sinh Vui Vẻ và Chúc Năm Mới Tốt Lành!
Καλά Χριστούγεννα! Ευτυχισμένο το Νέο Έτος!
Prettige kerstdagen en een Gelukkig Nieuwjaar!
Nollaig shona duit! Bliain úr faoi shéan is faoi mhaise duit!
Meri Kirihimete me ngā mihi o te tau hou ki a koutou katoa
С наступающим Новым Годом! С Рождеством Христовым!
Sinifesela uKhisimusi oMuhle noNyaka oMusha oNempumelelo!
Танд зул сарын баярын болон шинэ жилийн мэндийг хүргэе
Siniqwenelela Ikrisimesi Emnandi Nonyaka Omtsha Ozele Iintsikelelo
أجمل التهانئ بمناسبة الميلاد و حلول السنة الجديدة
本年もよろしくお願いいたします!
Schöni Wiänachtä, äs guets Nöis!
Selamat hari natal dan tahun baru!
Veselé vánoce a šťastný nový rok!
Juullimi ukiortaasamilu pilluaritsi
Buon Natale e felice anno nuovo!

Trainings 2012: Advanced iOS, Mobile Web Apps and Node.js

Adrian Kosmaczewski

2011-12-29

We are thrilled to announce new training sessions in Switzerland and South Africa this year:

- **Advanced iOS**: this training is for experienced iOS developers, looking to increase their knowledge with insider tips and tricks about the following subjects:
  - Building iOS 5.x universal applications, compatible with version 4.x of iOS.
  - Advanced user interface design for the iPhone and the iPad
  - Integrating your application with Twitter, Facebook and other social networking sites.
  - Performance enhancements using Core Foundation in your applications.
  - Using Instruments to find performance problems.
  - Major ninja-level optimizations.
  - 2D graphics and animation: Quartz and Core Animation.
  - Video, audio, the iPod library, AirPlay, etc.
- **Mobile Web App Development**: targeting web developers who want to take their existing knowledge to the next step:
  - Build mobile web applications such as to-do lists, a location-based social networking site and other apps using Sencha Touch[1], jQuery Mobile[2].
  - Learn how to integrate data from remote APIs in your mobile web apps, and how to organize your code and your architecture for the best results.
  - Package those applications using PhoneGap[3] and PhoneGap Build[4] and distribute them over the major mobile application marketplaces.
- **Web Development with Node.js**: this is our new training offering, for web developers who want to learn more about the latest and greatest web framework of the moment[5]:

[1] http://www.sencha.com/products/touch
[2] http://jquerymobile.com/
[3] http://phonegap.com/
[4] http://nodejs.org/
[5] http://nodejs.org/

- Learn how to create common web applications such as a blog, wiki, and management systems with backend databases, using the standard Node.js toolkit.
- Learn about other tools like npm[6], Express[7], Vows[8], MongoDB[9], Persistence.js[10] and more.

All trainings are 3 days long, and they include lunch and snacks during the day. All trainees will receive a ~100 page booklet written by Adrian with the most important aspects of the training, in PDF, ePub and .mobi (Kindle) formats. These trainings will take place throughout 2012 in Geneva and Zürich.

More information about dates and locations soon! We hope to see you soon!

Update: we forgot to mention that the trainings in Geneva will be given in French, while in Zürich they will be in English. We are also going to offer these in South Africa!

[6] http://npmjs.org/
[7] http://expressjs.com/
[8] http://vowsjs.org/
[9] http://www.mongodb.org/
[10] http://persistencejs.org/

Wordle 2012

Adrian Kosmaczewski

2012-01-09

Just like last year[1], here goes the Wordle cloud[2] generated with the most important tags in this blog for 2011!

Some subtle differences appear, don't you think? :)

[1] /blog/wordle/
[2] http://www.wordle.net/

Swiss App Awards: Don’t Forget!

Adrian Kosmaczewski

2012-01-10

The deadline of the Swiss App Awards[1] is approaching! Don’t forget to submit your application using the form[2] before January 15th!

The event will be held in Zürich, on January 27th, 2012. You can submit your application[3] (iOS, Android, Windows Mobile, web, any kind and in any language!) or just register as an attendee[4] on the website of the event[5]. You can also follow the official Twitter account[6] of the Swiss App Awards for more information.

We look forward to see your applications!

[7]

[1] http://swissappawards.ch/
[2] http://form.jotform.com/form/13471218230
[3] http://form.jotform.com/form/13471218230
[4] https://secure.trifork.com/swiss-app-awards-2012/registration/
[5] http://swissappawards.ch/
[6] http://twitter.com/SwissAppAwards
[7] http://swissappawards.ch/

Preferred iPad Apps

Adrian Kosmaczewski

2012-01-30

FlipBoard, Reeder, iA Writer, 1Password, Echofon, iWork (Pages, Numbers, Keynote), The Economist, OmniGraffle, InkPad, Adobe Ideas, iBooks, Kindle, Instagram, Instapaper, swissinfo, digital 2.0, Unzip, Typefaces, Prompt, FaceTime, Mirror, NYPL Biblion, iMovie, GarageBand, Keynote Remote, Zattoo HD, Aelios, TuneIn Radio, Planetary, VinylLove, Shazam, iDisk, Dropbox, Digits, GoodReader, DocsToGo, Articles, France24, CNN, La Nación Digital, OffMaps 2, Google Earth, Dictation, SBB Mobile, Penultimate, Elements, PCalc, Skype, Skype wifi, Hipmunk, Deep Green, Real Racing HD, ArtRage, Photoshop Express, HP Print.

Boom.

How we work

Adrian Kosmaczewski

2012-01-30

We have been in business for quite a few years, and we have found out several things that work and other that don't in our activities. We have had our share of successes and failures, as any other business; and we hope to have learnt something along the way.

This post will outline the recipes that we have applied to obtain the best results in the collaboration with our clients.

Principles

What are the principles that guide our work?

- To begin with, we do not work **for** our customers; we work **with** them. This means that we are both collaborating in our projects, in equal terms, with as many expectations as our clients have.
- Creating software is hard. If you "have a friend that can make your application for 100 dollars in 2 days", don't call us.
- If you are a "creative agency", and "your first class client" wants an application, and you try to push our rates down, "for the sake of our future collaboration", don't bother calling us. We're not interested.
- Similarly, if you are an agency and you are going to shield us and proxy (or "manage") all communications between us and the final customer, we are not interested. We only work in projects

where we have full access to the customer. We are not interested in other kinds of collaboration.
- We do not outsource projects[1]. Period.
- We do as much as we can to reduce our carbon footprint[2]. And you should, too.
- We do not sign NDAs. Really. Don't insist.

How we work

akosma software has a well defined workflow to help its clients bring their ideas to life; if you are into buzzwords, you could say that we use a mix of iterative and sequential approaches:

- We ask our clients for a single point-of-contact person, or a "Product Owner" as some methodologies call it; this person will be our sole communication proxy with the organization. We ask our clients to always forward questions and remarks through this person at any times. Doing so helps us work more efficiently and faster.
- We ask our clients for a well defined Statement of Work. We believe that the final product can only be as good as the Statement of Work provided. This document must provide as many details as possible about the final project. This helps us keep our costs low, it reduces misunderstandings, and helps us deliver features faster.
- For ad-hoc development projects, we usually work in "fixed time / fixed price" mode. We will provide you a cost and a deadline for your project, and we'll stick to it (that's why the Statement of Work is so important!) We can also work in "time and materials" mode if you need it.
- We try to manage project risks[3] from beginning to end. For that, we require your commitment, understanding and collaboration.
- We ask our clients to avoid email and phone as a communication and coordination medium. We use our project management site[4] instead, which features forums with threaded conversations, bug tracking and wikis. This allows us to centralize all the information of the project, as well as the communication with the client, in a single repository, to avoid losing information in crowded email inboxes.
- We provide feedback about our work continuously. Our clients can check the advancement of the project at any time in our project management site without even having to contact us. Everything we do is logged in the screens of the "Activity" panel, at any given time. This helps us reduce interruptions, which are costly, and also helps us increase customer satisfaction, because the client can always know what's going on.

[1] /blog/why-do-not-we-outsource-projects-overseas/
[2] /blog/reducing-the-carbon-footprint/
[3] /blog/risk-management-in-iphone-projects/
[4] http://projects.akosma.com

- We divide small projects (with a duration less than one month) in one-week long iterations, while longer or more complex projects are divided in two-weeks long iterations. At the end of each iteration, the client is able to test the application in their own devices and infrastructure, and we expect feedback from this. The clients' feedback is fundamental; without it, we will assume that everything is going according to plan and continue working in the next iteration.

Expectations

From each client that we agree to work with, we have as many expectations as you have.

- We expect you to provide us with clear explanations of what you need. We mean it. In written form. With diagrams. Color. Details. Everything.
- We expect your feedback, quickly, because every week we are going to use this feedback to make your product better.
- We expect you to take care of your project as we do. In particular, do not mess with the source code of what we provide to you: not only this breaks the warranty (yes, we provide one), but even worse, it breaks our trust in our relationship.
- We expect that you pay our invoices in time. As simple as that.
- We expect you to have a smile in your face, just like we want to have one when we work with you. That's why we do what we do.

If you would like to work with us, then just contact us![5]

[5] /about/

Determining Delegate Object Method Call Order in Objective-C with NSProxy

Adrian Kosmaczewski

2012-01-31

This is a guest post + code, wrote together with Joe D'Andrea[1] from LiquidJoe LLC[2]!

Many developers new to the iOS platform have trouble understanding the delegate architecture, in the sense that many other OO toolkits use properties to configure the characteristics of UI elements, instead of having a separate object doing the job.

When using delegates, the sequence of calls for each method is important: most Cocoa developers know that UITableView instances call delegate and data source methods roughly in this order:

- numberOfSections:
- numberOfRows:inSection:
- cellForRowAtIndexPath:
- configureCellAtIndexPath:

...mixing up in the middle some calls to headers and footers and cell sizes as well. Which prompts the following question: **what is really going on behind the scenes? Can we know exactly the order in which those delegate methods are called?**

Here's a small project that uses all the delegate and datasource methods in the same project, and that uses NSLog to show which calls happen first, and when.

Delegate Methods

But first, a quick recap on delegate methods in Objective-C.

In many (if not most) object oriented toolkits, properties tend to be used when setting up an object such as a table view, for instance in .NET, as shown in this example from Microsoft[3]

```
<%@ Page language="c#" %>
<%@ Import Namespace="System.Data" %>
```

[1] https://github.com/jdandrea

[2] http://www.liquidjoe.biz/

[3] http://support.microsoft.com/kb/307860

```
<%@ Import Namespace="System.Data.SqlClient" %>

<script runat="server">
void Page_Load(Object sender, EventArgs e)
{
    SqlConnection cnn = new
        SqlConnection("server=(local);database=pubs;Integrated Security=SSPI");
    SqlDataAdapter da = new SqlDataAdapter("select * from authors", cnn);
    DataSet ds = new DataSet();
    da.Fill(ds, "authors");
    Repeater1.DataSource = ds.Tables["authors"];
    Repeater1.DataBind();
}
</script>
<html>
<body>
    <form id="WebForm2" method="post" runat="server">
        <asp:Repeater id="Repeater1" runat="server">
            <ItemTemplate>
            <%# DataBinder.Eval(Container.DataItem,"au_id") %><br>
            </ItemTemplate>
        </asp:Repeater>
    </form>
</body>
</html>
```

In the code above (a classic ASP.NET page) the `Repeater1` instance (of the `Repeater` class, which is a view component) is "bound" (a classic example of "databinding") to an instance of a `DataSet`. In this case, we just call the `.DataBind()` method on the repeater object, and the view component is filled with the data inside.

The advantage of this approach is that it is simpler from the point of view of the developer; a couple of lines of code are usually enough. However, this design tends to break the MVC model, because it creates a direct dependency between the model and the view component. This coupling makes it hard to modify the model of the application without breaking the view layer.

In Objective-C this pattern is seldom used. Instead, an object will opt to let someone else do the job for them. That other object is known as the **delegate object**, and the methods to be implemented (sometimes optionally) are referred to as a protocol. In this model, there is a controller object that acts as intermediate between the view and the model, effectively creating a decoupled MVC architecture.

In Objective-C, an object tells the compiler that it implements a protocol using this syntax:

```
@interface MyAppDelegate : NSObject <UIApplicationDelegate>
@end
```

The inclusion of `<UIApplicationDelegate>` signals that this class

conforms to the UIApplicationDelegate protocol. Meanwhile, in UIApplication.h, you will see this:

```
@protocol UIApplicationDelegate <NSObject>
@optional

- (void)applicationDidFinishLaunching:(UIApplication *)application;
- (BOOL)application:(UIApplication *)application didFinishLaunchingWithOptions

...
@end
```

Thus, MyAppDelegate.m may optionally respond to `-applicationDidFinishLaunching:` and `-application:didFinishLaunchingWithOptions:`. (In this particular case, be aware that the former method is used by apps prior to iOS 3.0, and that the latter method should now be used instead.)

More about Delegates

It is up to the class in question to determine when to call a delegate method, and in what order. This order is of particular interest when it comes to iOS Table Views. In fact, go ahead and press Cmd-Shift-O in Xcode right now. Search for and open UITableView.h.

Look inside. There's a lot to take in here! In fact, that's a good idea - take a moment to browse this header. (Go ahead. We'll wait.) A Cocoa developer can write apps and never look once inside these header files. To borrow (abuse?) a classic Apple mantra, we code different. We want to get our hands dirty and look inside at the engine ... or at least as much of it as possible. The instance variables are out of our reach, but there are plenty of other things to take in.

UITableView is unusual in that it declares two protocols, whereas most classes you come across will only declare one.

The UITableViewDelegate is responsible for mediating all table cell display and behavior. A quick look at the protocol makes this abundantly clear. There are methods covering custom display, variable height, headers and footers, accessories, selection, editing, reordering, and indentation.

```
@protocol UITableViewDelegate <NSObject, UIScrollViewDelegate>
```

UITableViewDataSource represents the data model object. While it supplies no information about appearance – of the cells or otherwise – this doesn't mean it avoids cells altogether. There are methods covering cell view creation (and, more importantly, reuse), the number of sections and rows, titles for headers and footers, index titles along the right-hand side, plus cell editing, insertion, deletion, and moving.

In this protocol, two methods are required:

```
- (NSInteger)tableView:(UITableView *)table numberOfRowsInSection:(NSInteger)se
- (UITableViewCell *)tableView:(UITableView *)tableView cellForRowAtIndexPath:(
```

The rest are optional. That's right, even -numberOfSectionsInTableView:, which you almost always see implemented like so:

```
- (NSInteger)numberOfSectionsInTableView:(UITableView *)tableView
{
    return 1;
}
```

However, the header tells us that the default value is 1 if this is not implemented! If all you have is one section, save yourself some extra coding and leave it out. Aren't well-documented header files great?

Now there has to be some degree of bootstrapping involved with setting up these delegates. This is accomplished via properties:

```
@property (nonatomic,assign) id <UITableViewDataSource> dataSource;
@property (nonatomic,assign) id <UITableViewDelegate>   delegate;
```

... and usually set to self:

```
// Set the table view and data source delegates:
myTableView.delegate = self;
myTableView.dataSource = self;
```

If you use a UITableViewController, this is done for you automagically, in addition to several other niceties. (We heart UITableViewController. Use it early. Use it often!)

Delegate Method Call Order

Great. So we have two delegate objects that can receive almost 30 method calls combined. Not five. Not ten. Thirty. Helpful as the delegate design pattern is, it's easy to get confused knowing which method will be called at which time. Thus, it can also be instructive to know what order these methods are called.

You can probably guess a few of them right off the bat. For instance, -tableView:numberOfSections: is likely called before -tableView:numberOfRows:inSection:. It's not always so clear cut, though.

So how do we trace Objective-C messages anyway? One possibility is to set up an Xcode breakpoint and declare this as an action (borrowed from StackOverflow[4]):

```
while 1
printf "[%s %s]", (char *)object_getClassName($r0), (char *) $r1
c
end
```

Hmm ... maybe not.

[4] http://stackoverflow.com/questions/1029962/nsobjcmessageloggingenabled-with-iphone-3-0

Is it possible to trace Objective-C messages en masse? As a matter of fact, it is[5]:

On the iPhone Simulator, you can compile with the `NSObjCMessageLoggingEnabled` Foundation environment variable set to YES, then look in `/tmp/msgSends-<pid>` (where is your running app's process id).

You'll also get a deluge of messages, accent on deluge.

What else? You could use (the undocumented[6]) `-instrumentObjcMessageSends:` to toggle logging on and off.

Hmm. This isn't looking so good. "There's got to be a better way!"

We can certainly drop breakpoints in each of our delegate methods and trace through them, but wouldn't it be nice to just log the call order of our class delegates in one swell foop, without resorting to undocumented or cumbersome means?

Using an NSProxy

Of course there is. That's why we wrote this blog post, after all, so we're going to show you how it's done. Best of all, you can use this technique for any class. Roll up your sleeves and let's get started.

[This code is adapted from http://blog.jayway.com/2009/03/06/proxy-based-aop-for-cocoa-touch/]

Remember, we're operating under the notion that you know your way around Xcode and have already built a project or two! In Xcode, create a new "Master-Detail" iOS Application called DelegateOrder. We're going to use a PTV prefix for all of our classes (why? Long story :)

Thanks to the wonders of modern template technology, our root view controller happens to be a table view. How convenient!

Time to create a new class. Right-click on the Classes group in Xcode, and choose New File... from the popup menu. Add an Objective-C Cocoa Touch subclass of NSObject, and name it PTVControllerProxy. Open PTVControllerProxy.h and change it to look like this:

```
@interface PTVControllerProxy : NSProxy

+ (id<UITableViewDataSource, UITableViewDelegate>)proxyWithTableViewController
- (id)initWithTableViewController:(UITableViewController *)controller;

@end
```

Notice anything different?

We have not so secretly replaced the NSObject Xcode usually serves with a dark, sparkling NSProxy object. (Footnote: This is a cultural

[5] http://www.dribin.org/dave/blog/archives/2006/04/22/tracing_objc/
[6] http://developer.apple.com/technotes/tn2004/tn2124.html

reference to a famous Folgers coffee commercial that ran in the US in the '80s. Not sure if it aired worldwide though!) Like NSObject, NSProxy is a root class, except it also conforms to NSObject, plus it adds a few methods of its own. Here's the header:

```
#import <Foundation/NSObject.h>

@class NSMethodSignature, NSInvocation;

@interface NSProxy <NSObject> {
    Class   isa;
}

+ (id)alloc;
+ (id)allocWithZone:(NSZone *)zone;
+ (Class)class;

- (void)forwardInvocation:(NSInvocation *)invocation;
- (NSMethodSignature *)methodSignatureForSelector:(SEL)sel;
- (void)dealloc;
- (void)finalize;
- (NSString *)description;
+ (BOOL)respondsToSelector:(SEL)aSelector;

@end
```

NSProxy objects act as stand-ins for other objects. In our case, we'll use PTVControllerProxy as our table view's dataSource and delegate. It will also know about our actual Table View Controller (and table view) so that it can pass messages down the line.

The next item of interest is our private instance variable, _controller, a pointer to a UITableViewController object.

Following the Coding Guidelines for Cocoa, we always make our instance variables private. We also add a leading underscore – just like Apple does with their own headers – so as to distinguish them from properties. This further enforces class interaction through properties and methods. Meanwhile, on the implementation side, sometimes you want to access the instance variable. Other times, you need to use the property. Underscores make it clear which is which.

In this case, we're not creating a property … at least not one that's publicly available. More on that later.

We've also defined one class method and one instance method:

```
+ (id<UITableViewDataSource, UITableViewDelegate>)proxyWithTableViewController
```

Our class method takes a UITableViewController and returns an object that conforms to both the Table View Data Source and Delegate protocols.

```
- (id)initWithTableViewController:(UITableViewController *)controller;
```

Our designated initializer method (called by the class method) does the deed of setting our controller property.

Did we mean _controller instance variable? No, we meant property. Now press Cmd-Option-Up-Arrow to switch to the implementation file and make it look like this:

```
#import "PTVControllerProxy.h"

@interface PTVControllerProxy ()

@property (nonatomic, assign) UITableViewController *controller;

- (void)logInvocation:(NSInvocation *)invocation;

@end


@implementation PTVControllerProxy

@synthesize controller = _controller;

@end
```

Ah-ha! So that's where the property went.

We have included yet another interface declaration for PTVControllerProxy, and this time we're extending it beyond what we originally declared in the header. The parentheses after @interface PTVControllerProxy signify that this is a class extension. In this case, we've declared a property and a method that isn't exposed to the outside world. It's not private per se, but it's not visible in the header either. Next, in the implementation, we've synthesized a property named controller, which is represented by the instance variable named _controller.

Due to the architecture of Objective-C, you could make a few end-runs around this and still get at the property and methods. You just have to know how they're defined. Be careful of doing this with Apple's classes though. Them's grounds for App Store rejection!

You can even override extensions previously defined in the header, making this technique quite handy for exposing readwrite properties as readonly. Read more about Categories and Extensions in The Objective-C Programming Language.

We'll discuss `-logInvocation:` in a moment, but you probably have a good idea of what it will do just by looking at the name.

Time to add our methods. Heads up: Some of these may – or will – be a bit unfamiliar. All we are doing here is implementing the minimum number of methods to handle proxying for our table view controller, and then some.

The following should be added between @synthesize and @end in our

class implementation. First, we add our previously defined initializers, and our lone NSObject method:

```
#pragma mark - Initializers

+ (id<UITableViewDataSource, UITableViewDelegate>)proxyWithTableViewController
{
    return [[[[self class] alloc] initWithTableViewController:controller] autor
}

- (id)initWithTableViewController:(UITableViewController *)controller
{
    self.controller = controller;
    return self;
}

#pragma mark - NSObject

- (void)dealloc
{
    [_controller release];
    [super dealloc];
}
```

Our class method takes a table view controller and sends it along to the designated initializer, which in turn assigns it to the property controller, and returns itself. As there is no superclass to worry about, there is no use of `[super init]` here.

There's one more benefit to our use of private instance variables beginning with underscores. Our method parameters do not conflict with like-named instance variables! Look at this line again:

```
self.controller = controller;
```

This dot notation is 100% equivalent to:

```
[self setController:controller];
```

Very clean, consistent, and easy to read. Contrast this with the scenario where our instance and method variable named matched. We would have to code methods like this:

```
- (id)initWithTableViewController:(UITableViewController *)theController
{
    self.controller = theController;
    return self;
}
```

It should also be pointed out that these method names are not haphazard. Apple does in fact provide a method (pun intended) for naming methods! The previously mentioned Coding Guidelines for Cocoa tells all. Take the time to name your instance variables, properties, and methods according to the guidelines. It's worth the effort!

The dealloc method is also standard-issue. Notice that we work with the instance variable at this point. After dealloc, the show's over for this class, so we don't even bother setting _controller to nil. In other cases, you would likely want to clear it, using the property to take advantage of its retain/release smarts in one step:

```
// This setter (-setController:) releases _controller for us!
self.controller = nil;
```

Continuing with our NSObject methods, add the following:

```
- (BOOL)isKindOfClass:(Class)aClass;
{
    return [self.controller isKindOfClass:aClass];
}

- (BOOL)conformsToProtocol:(Protocol *)aProtocol;
{
    return [self.controller conformsToProtocol:aProtocol];
}

- (BOOL)respondsToSelector:(SEL)aSelector;
{
    return [self.controller respondsToSelector:aSelector];
}
```

Be careful here. There are class methods with the same signature like NSProxy's +respondsToSelector:! We're responding to NSObject's protocol methods instead, passing (well, proxying) the parameter over to our table view controller. This takes care of establishing our proxy as having the same class, protocol conformance, and selector response. That, in turn, assures us an opportunity to tap in to the deluge of table view delegate traffic.

Now, so we just discussed how you can avoid naming collisions between method parameters and instance variables, and here we have aClass, aProtocol, and aSelector! This leads us to an important maxim: "You have to know the rules before you can break them." Use discretion! Here, we're honoring the naming convention established by the authors of NSObject.h, in particular the NSObject protocol.

As it turns out, the Protocol and Base Class declarations of `-conformsToProtocol:` are ever so slightly different:

```
// Protocol
- (BOOL)conformsToProtocol:(Protocol *)aProtocol;

// Base Class
+ (BOOL)conformsToProtocol:(Protocol *)protocol;
```

Go figure. (It's OK. We still love Cocoa. Tasty Cocoa ...)

Let's add two more, slightly longer methods, this time from NSProxy's protocol:

```
#pragma mark - NSProxy

- (NSMethodSignature *)methodSignatureForSelector:(SEL)aSelector
{
    if ([self.controller respondsToSelector:aSelector])
    {
        return [self.controller methodSignatureForSelector:aSelector];
    }
    else
    {
        return [super methodSignatureForSelector:aSelector];
    }
}
```

Method signatures help us forward along messages we otherwise wouldn't respond to. Here, the message is either bound for our table view controller or the superclass.

```
- (void)forwardInvocation:(NSInvocation *)invocation
{
    SEL selector = [invocation selector];
    if ([self.controller respondsToSelector:selector])
    {
        [self logInvocation:invocation];
        [invocation setTarget:self.controller];
        [invocation invoke];
    }
}
```

`-forwardInvocation:` is used by subclasses to send Objective-C messages-as-objects (or Invocations) to other objects. For the proxy, it's just a matter of passing the message along to whatever object we aim to represent. That object, of course, is the table view controller.

First, we ensure the controller responses to the selector represented by the invocation. Then, just before we dispatch the invocation using -invoke, we call - ta-dah - our private -loginvocation: method. With that, we can now log every single solitary table view controller message that comes our way, all from one place.

Oh, and look, it's invocation and not anInvocation. (You're not going to let us live this one down, are you.)

At last, we come to the coup de grace of our controller proxy class. Add this to the implementation:

```
#pragma mark - Private

- (void)logInvocation:(NSInvocation *)invocation
{
    SEL selector = [invocation selector];
    NSString *currentMethod = NSStringFromSelector(selector);
    NSString *argument = @"";
```

```
    NSMethodSignature *methodSignature = [invocation methodSignature];
    NSInteger argCount = [methodSignature numberOfArguments];

    for (NSInteger index = 3; index < argCount; ++index)
    {
        const char *argType = [methodSignature getArgumentTypeAtIndex:index];

        if (strcmp(argType, "@") == 0)
        {
            id object = nil;
            [invocation getArgument:&object atIndex:index];
            if ([object isKindOfClass:[NSIndexPath class]])
            {
                NSIndexPath *indexPath = (NSIndexPath *)object;
                argument = [NSString stringWithFormat:@"{%d, %d}", indexPath.se
            }
        }
        else if (strcmp(argType, "i") == 0)
        {
            NSInteger section;
            [invocation getArgument:&section atIndex:index];
            argument = [NSString stringWithFormat:@"%d", section];
        }
    }
    NSLog(@"%@%@", currentMethod, argument);
}
```

OK! A lot to take in here. Let's talk through it.

Just like in `-forwardInvocation:`, we grab our selector, or in this case our delegate message. In preparation for logging, we turn it into a string called `currentMethod`. This gives us, for example, @"tableView:cellForRowAtIndexPath:" - ready for logging.

Perhaps that would be enough, but we can do better. NSInvocation contains a NSMethodSignature object with all the arguments, so let's grab a few of those while we're at it.

If you were to look at the method signature's arguments, you would soon discover the first two arguments are always claimed by a few behind-the-scenes players, (id)self and (SEL)_cmd. Argument indices are zero-based, so that places us at index 2 for the start of our delegate methods. Meanwhile, following best practices for delegate method conventions, the table view is always passed back as the first argument, in this case index 2. Thus we want to start at index 3, which should explain this loop construct:

```
for (NSInteger index = 3; index < argCount; ++index)
{
    const char *argType = [methodSignature getArgumentTypeAtIndex:index];
...
```

Next, we look at the argument type, returned as constant C string representing an Objective C argument type. “@” represents an object, and “i” represents an integer. Easy enough.

If we have an object, we check to see if it’s of class NSIndexPath. If so, good! We grab the index path’s row and section, and use that in the log message. If we have an integer, we simply grab the integer value instead.

What if we have no such argument, as in -numberOfSectionsInTableView? No worries, we’ll still log the message. We just won’t have any of these parameters to go along with it.

Finally, we log our method and our (possible) lone argument. At last, the controller proxy is complete.

We now turn our attention to our table view controller, represented here by PTVRootViewController. Open the header file first, and change it to look like this:

```
#import <UIKit/UIKit.h>

@interface PTVRootViewController : UITableViewController
{
@private
    id<UITableViewDataSource, UITableViewDelegate> _proxy;
}

@end
```

We’ve merely added a private instance variable for the proxy object. Notice that there is no publicly exposed property. We will only be using the proxy within the confines of the class.

Now switch to the implementation file and … remove everything. Nothing wrong with the pre-fab code here. We’re just going to start over. Add this for starters:

```
#import "PTVRootViewController.h"
#import "PTVControllerProxy.h"

static NSInteger SECTION_COUNT = 25;
static NSInteger ROW_COUNT = 5;
```

We first import our class header, as well as the proxy header. Next, because we’re testing out the proxy, we can keep the data source rather sparse and simple, but we still want a good number of rows and sections to work with. We define a few constants to this effect, giving us 25 table view sections, each one with five rows.

Time for another private extension:

```
@interface PTVRootViewController ()

@property (nonatomic, retain) id<UITableViewDataSource, UITableViewDelegate> pr
```

```
@end


@implementation PTVRootViewController

@synthesize proxy = _proxy;

@end
```

There it is, our internal proxy property, and a bare-bones implementation, complete with synthesized methods. Next up is our requisite NSObject method:

```
#pragma mark - NSObject

- (void)dealloc
{
    [_proxy release];
    [super dealloc];
}
```

Nothing earth-shattering there. Next up, a bunch of UIViewController methods:

```
#pragma mark - UIViewController

- (void)viewDidLoad
{
    [super viewDidLoad];

    self.proxy = [PTVControllerProxy proxyWithTableViewController:self];
    self.tableView.dataSource = self.proxy;
    self.tableView.delegate = self.proxy;

    self.title = @"Delegate Call Order";
    self.navigationItem.rightBarButtonItem = self.editButtonItem;
}
```

In -viewDidLoad, we create and connect the proxy object. We also take a moment to set the associated navigation bar's title and add a stock edit button on the right side.

```
- (void)viewDidUnload
{
}

- (void)viewWillAppear:(BOOL)animated
{
    [super viewWillAppear:animated];
}

- (void)viewDidAppear:(BOOL)animated
```

```
{
    [super viewDidAppear:animated];
}

- (void)viewWillDisappear:(BOOL)animated
{
    [super viewWillDisappear:animated];
}

- (void)viewDidDisappear:(BOOL)animated
{
    [super viewDidDisappear:animated];
}

- (BOOL)shouldAutorotateToInterfaceOrientation:(UIInterfaceOrientation)interfac
{
    return (interfaceOrientation == UIInterfaceOrientationPortrait);
}
```

We'll disable autorotation for now, so we only respond to portrait orientation requests.

```
- (void)didReceiveMemoryWarning
{
    [super didReceiveMemoryWarning];
}
```

At last! Time to add the delegate methods. All of them. That's right, **we will respond to every single one**. In doing so, the proxy will get a chance to work its logging magic. Add the following for starters:

```
#pragma mark - UITableViewDataSource

- (NSInteger)numberOfSectionsInTableView:(UITableView *)tableView
{
    return SECTION_COUNT;
}

- (NSArray *)sectionIndexTitlesForTableView:(UITableView *)tableView
{
    NSMutableArray *array = [NSMutableArray arrayWithCapacity:SECTION_COUNT];
    for (NSInteger index = 0; index < SECTION_COUNT; ++index)
    {
        NSString *sectionTitle = [NSString stringWithFormat:@"%d", index];
        [array addObject:sectionTitle];
    }
    return array;
}
```

Our table view has SECTION_COUNT sections. The section index titles (along the right side of the table view) will be represented by our section numbers 0 through 24. This will make it easy to see the effects

of changing our vantage point.

Let's add a few more:

```
- (BOOL)tableView:(UITableView *)tableView canEditRowAtIndexPath:(NSIndexPath *
{
    return YES;
}

- (BOOL)tableView:(UITableView *)tableView canMoveRowAtIndexPath:(NSIndexPath *
{
    return YES;
}
```

We allow editing and moving of all rows.

```
- (UITableViewCell *)tableView:(UITableView *)tableView cellForRowAtIndexPath:(
{
    static NSString *CellIdentifier = @"Cell";

    UITableViewCell *cell = [tableView dequeueReusableCellWithIdentifier:CellId
    if (cell == nil)
    {
        cell = [[[UITableViewCell alloc] initWithStyle:UITableViewCellStyleDefa
    }

    cell.textLabel.text = [NSString stringWithFormat:@"Section %d, row %d", ind

    return cell;
}
```

Here is where the cell generation and reuse takes place. Again, nothing surprising here. We take advantage of our table view's cell queue and either dequeue or create a cell, setting its text label to the current cell's section and row.

```
- (void)tableView:(UITableView *)tableView commitEditingStyle:(UITableViewCellE
{
    if (editingStyle == UITableViewCellEditingStyleDelete)
    {
        [tableView deleteRowsAtIndexPaths:[NSArray arrayWithObject:indexPath] w
    }
    else if (editingStyle == UITableViewCellEditingStyleInsert)
    {
    }
}
```

Since we're allowing editing (and by editing we mean deletion of cells, not insertion), we'll respond to this method as well, deleting the row indicated by the indexPath, fading it out as it goes.

```
- (void)tableView:(UITableView *)tableView moveRowAtIndexPath:(NSIndexPath *)fr
{
}
```

```
- (NSInteger)tableView:(UITableView *)tableView numberOfRowsInSection:(NSIntege
{
    return ROW_COUNT;
}

- (NSInteger)tableView:(UITableView *)tableView sectionForSectionIndexTitle:(NS
{
    return [title intValue];
}
```

We respond to `-tableView:moveRowAtIndexPath:toIndexPath:` as well. Since we aren't manipulating any behind-the-scenes data, we don't need to do anything extra. We also offer up `ROW_COUNT` rows in each section. Next, we take advantage of our section index titles (which are just numbers represented as strings), converting them back to integers as needed. index is meant to be used with our index title array, which of course we aren't keeping track of. Then again, it's simple enough that we don't have to. Each index title maps directly to a like-numbered section in the table view, so it's as simple as returning each title's integer value.

Two more data source methods and we're more than halfway there.

```
- (NSString *)tableView:(UITableView *)tableView titleForFooterInSection:(NSInt
{
    return [NSString stringWithFormat:@"Footer for section %d", section];
}

- (NSString *)tableView:(UITableView *)tableView titleForHeaderInSection:(NSInt
{
    return [NSString stringWithFormat:@"Header for section %d", section];
}
```

Easy peasy, right? Good! Now for the table view delegate methods. These are even easier:

```
#pragma mark - UITableViewDelegate

- (void)tableView:(UITableView *)tableView accessoryButtonTappedForRowWithIndex
{
}

- (void)tableView:(UITableView *)tableView didDeselectRowAtIndexPath:(NSIndexPa
{
}

- (void)tableView:(UITableView *)tableView didEndEditingRowAtIndexPath:(NSIndex
{
}

- (void)tableView:(UITableView *)tableView didSelectRowAtIndexPath:(NSIndexPath
```

```
{
    [self.tableView deselectRowAtIndexPath:indexPath animated:YES];
}

- (UITableViewCellEditingStyle)tableView:(UITableView *)tableView editingStyleF
{
    return UITableViewCellEditingStyleDelete;
}
```

A few more empty methods, again just to get the benefit of logging. We will also deselect each row in response to it being selected. Next, we assure that the editing style of each row supports deletion.

Now we'll handle header, footer, and row sizing, plus indentation:

```
- (CGFloat)tableView:(UITableView *)tableView heightForFooterInSection:(NSInteg
{
    return 20.0f;
}

- (CGFloat)tableView:(UITableView *)tableView heightForHeaderInSection:(NSInteg
{
    return 20.0f;
}

- (CGFloat)tableView:(UITableView *)tableView heightForRowAtIndexPath:(NSIndexP
{
    return 44.0f;
}

- (NSInteger)tableView:(UITableView *)tableView indentationLevelForRowAtIndexPa
{
    return 0;
}

- (BOOL)tableView:(UITableView *)tableView shouldIndentWhileEditingRowAtIndexPa
{
    return YES;
}
```

Returning YES for `-tableView:shouldIndentWhileEditingRowAtIndexPath:` allows the background of the edited row to be indented. This allows the cell contents to shift and more gracefully accomodate the Delete button appearing on the right-hand side.

```
- (NSIndexPath *)tableView:(UITableView *)tableView targetIndexPathForMoveFromR
{
    return proposedDestinationIndexPath;
}
```

We won't do anything unusual with proposed moving of cells, so we return the proposed destination index path.

```
- (NSString *)tableView:(UITableView *)tableView titleForDeleteConfirmationButt
{
    return @"Delete";
}

- (UIView *)tableView:(UITableView *)tableView viewForFooterInSection:(NSIntege
{
    return nil;
}

- (UIView *)tableView:(UITableView *)tableView viewForHeaderInSection:(NSIntege
{
    return nil;
}
```

The delete confirmation button will remain @“Delete”, and we won’t be using views for the section headers or footers. Still, we need to implement these methods if we are to see them logged!

Four more methods to go, and our implementation is finished:

```
- (void)tableView:(UITableView *)tableView willBeginEditingRowAtIndexPath:(NSIn
{
}

- (NSIndexPath *)tableView:(UITableView *)tableView willDeselectRowAtIndexPath:
{
    return indexPath;
}

- (void)tableView:(UITableView *)tableView willDisplayCell:(UITableViewCell *)c
{
}

- (NSIndexPath *)tableView:(UITableView *)tableView willSelectRowAtIndexPath:(N
{
    return indexPath;
}
```

Executing the Code

Congratulations! Let’s take it for a spin in the Simulator. Before building your project, bring up the debugger console window by typing Shift-Cmd-R or selecting Run > Console from the menu. Move the console off to the side so that you can see what happens next. You might even want to venture a guess as to which messages and parameters you’ll see first. Place your bets!

Now Build and Run your project using the Debug configuration. The Simulator will launch and the console will soon be filled with a bunch of log messages. Do not use the Simulator just yet. Let’s focus on the

console.

Thankfully, they will only be concerning our table view controller delegates. (Otherwise, we would be positively buried with messages. You think this is a lot of messages, you should see what Objective-C deals with in a typical app's lifetime!)

Let's start at the top and see what we have here. (We have removed the timestamp and App specific info. Your log lines will be a bit lengthier by comparison. You might want to make the console window a bit wider to help keep the lines from wrapping.)

```
numberOfSectionsInTableView:
numberOfSectionsInTableView:
```

The first thing we're asked for is the number of sections in our table view … and we're asked for this information twice. (Your guess is as good as ours.)

Remember, we have 24 sections in our table view.

```
tableView:viewForHeaderInSection:24
tableView:titleForHeaderInSection:24
tableView:heightForHeaderInSection:24
tableView:heightForHeaderInSection:24
tableView:viewForFooterInSection:24
tableView:titleForFooterInSection:24
tableView:heightForFooterInSection:24
tableView:heightForFooterInSection:24
```

Next, we see a bunch of messages pertaining to one section, in this case section 24. First the headers, then the footers. Here we have two more examples of a twice-called method in `-tableView:heightForHeaderInSection:` and `-tableView:heightForFooterInSection:`.

So the order so far is as follows (eliminating duplicate invocations):

```
numberOfSectionsInTableView:
```

Then, for each section, the header and footer information:

```
tableView:viewForHeaderInSection:
tableView:titleForHeaderInSection:
tableView:heightForHeaderInSection:
tableView:viewForFooterInSection:
tableView:titleForFooterInSection:
tableView:heightForFooterInSection:
```

You can imagine that the title is not needed (and the relevant method not called) if a view is returned, so it makes sense that the view is asked for first.

Armed with our section count, the table view turns its attention to the rows, starting with … the last one?

```
tableView:numberOfRowsInSection:24
tableView:heightForRowAtIndexPath:{24, 0}
```

```
tableView:heightForRowAtIndexPath:{24, 1}
tableView:heightForRowAtIndexPath:{24, 2}
tableView:heightForRowAtIndexPath:{24, 3}
tableView:heightForRowAtIndexPath:{24, 4}
```

We have 5 rows in each section, which explains the index paths from {24, 0} through {24, 4}. The height is returned for each. So far, so good.

Now it would appear as if we're starting at the end of the list and working our way backward, but the next thing you see is this:

```
tableView:viewForHeaderInSection:0
tableView:titleForHeaderInSection:0
tableView:heightForHeaderInSection:0
tableView:heightForHeaderInSection:0
tableView:viewForFooterInSection:0
tableView:titleForFooterInSection:0
tableView:heightForFooterInSection:0
tableView:heightForFooterInSection:0
tableView:numberOfRowsInSection:0
tableView:heightForRowAtIndexPath:{0, 0}
tableView:heightForRowAtIndexPath:{0, 1}
tableView:heightForRowAtIndexPath:{0, 2}
tableView:heightForRowAtIndexPath:{0, 3}
tableView:heightForRowAtIndexPath:{0, 4}
```

Is UITableView darting back and forth between the beginning and the end? (Answer: No. The next section it inquires about is section 1, all the way up to 23.) In fact, if you recompile this with only three sections, numbered 0 through 2, you'll get a similar outcome with the section order: 2, 0, and 1.

So what's happening here? Why does UITableView need to know about the last section first? [Note: Unknown! Any insights?]

```
sectionIndexTitlesForTableView:
```

Now that UITableView knows all the row heights, you shouldn't see those called again unless the table is reloaded. That's a good thing too because calculating those row heights can take some time, depending on the situation.

```
tableView:cellForRowAtIndexPath:{0, 0}
tableView:indentationLevelForRowAtIndexPath:{0, 0}
tableView:canEditRowAtIndexPath:{0, 0}
tableView:willDisplayCell:forRowAtIndexPath:{0, 0}
```

Next up are a series of four messages for each visible row, as well as those just out of view above and below. Since we're at the start of the content area, we begin with index path {0, 0} and the messages are sent all the way through index path {1, 3}. The familiar `-tableView:cellForRowAtIndexPath:` is called first, followed by a request for the indentation level, and a check to see if the row is editable.

Last comes `-tableView:willDisplayCell:forRowAtIndexPath:`. If you've read the documentation, you know this is your last chance to make final adjustments before a cell is displayed ... and now we have proof!

```
tableView:viewForHeaderInSection:0
tableView:titleForHeaderInSection:0
tableView:viewForFooterInSection:0
tableView:titleForFooterInSection:0
```

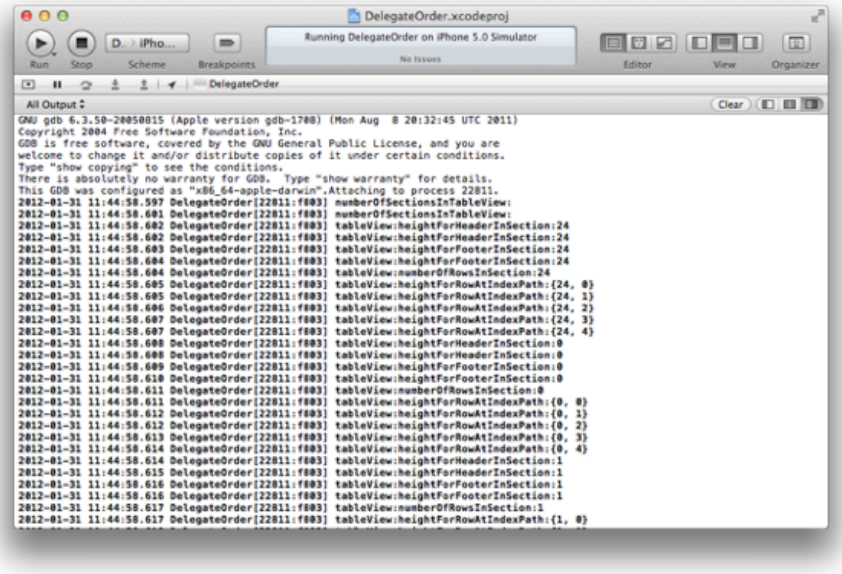
Finally, we have a set of four different messages for the view (or, failing that, the title) for the header in each visible section. You probably know that headers are always visible, even when all of a given section's rows are not. This holds true for section footers as well, only these are kept visible on the bottom of the view instead of the top.

In the Debugger console, press return a few times to add some blank space (or just press Clear Log in the toolbar). Now let's try and scroll another screenful of rows into view and see what happens. Click and drag the bottom-most row up to the top until Section 2, row 4 is visible. Here's what we get. Note that we have added blank lines in between each logical group of method invocations.

```
tableView:viewForHeaderInSection:0
tableView:titleForHeaderInSection:0

tableView:cellForRowAtIndexPath:{1, 4}
tableView:indentationLevelForRowAtIndexPath:{1, 4}
tableView:canEditRowAtIndexPath:{1, 4}
tableView:willDisplayCell:forRowAtIndexPath:{1, 4}

tableView:viewForFooterInSection:1
tableView:titleForFooterInSection:1

tableView:viewForHeaderInSection:2
tableView:titleForHeaderInSection:2

tableView:cellForRowAtIndexPath:{2, 0}
tableView:indentationLevelForRowAtIndexPath:{2, 0}
tableView:canEditRowAtIndexPath:{2, 0}
tableView:willDisplayCell:forRowAtIndexPath:{2, 0}

tableView:viewForFooterInSection:2
tableView:titleForFooterInSection:2

tableView:cellForRowAtIndexPath:{2, 1}
tableView:indentationLevelForRowAtIndexPath:{2, 1}
tableView:canEditRowAtIndexPath:{2, 1}
tableView:willDisplayCell:forRowAtIndexPath:{2, 1}

tableView:cellForRowAtIndexPath:{2, 2}
tableView:indentationLevelForRowAtIndexPath:{2, 2}
```

```
tableView:canEditRowAtIndexPath:{2, 2}
tableView:willDisplayCell:forRowAtIndexPath:{2, 2}

tableView:cellForRowAtIndexPath:{2, 3}
tableView:indentationLevelForRowAtIndexPath:{2, 3}
tableView:canEditRowAtIndexPath:{2, 3}
tableView:willDisplayCell:forRowAtIndexPath:{2, 3}

tableView:viewForHeaderInSection:1
tableView:titleForHeaderInSection:1

tableView:cellForRowAtIndexPath:{2, 4}
tableView:indentationLevelForRowAtIndexPath:{2, 4}
tableView:canEditRowAtIndexPath:{2, 4}
tableView:willDisplayCell:forRowAtIndexPath:{2, 4}
```

Of particular interest here are the header and footer related methods. They appear to be invoked in a rather odd pattern, until you examine the order that each header and footer appears on screen. Then it makes perfect sense. Interestingly enough, now that we're on the move and have stopped at a definitive spot, we never see index path {3, 0} on the radar, which is just out of view, past the bottom.



What about editing? Add some blank space in your debugger, then tap the edit button.

```
tableView:canEditRowAtIndexPath:{1, 1}
tableView:editingStyleForRowAtIndexPath:{1, 1}
tableView:shouldIndentWhileEditingRowAtIndexPath:{1, 1}
tableView:canMoveRowAtIndexPath:{1, 1}
```

We've already seen -tableView:canEditRowAtIndexPath: invoked, just before our last chance to adjust the cell prior to display. Now it's the first message in the group, followed by a few additional requests. The editing style is sought, followed by a request to indent the row while editing, and rounded out by an inquiry to see if the row can be moved.

Here's another observation. Notice how index path {1, 1} seems to be out of view in our example, yet we see it being called here. This is repeated through index path {2, 4} which is just in view, but not index path {3, 0} which is just out of view once again. Why?

There is a good reason for this behavior. When a header and footer are covering up rows within their related section (that is, when the header or footer is pegged to the top or bottom of the visible part of the view), look for a little bit of transparency, letting the underlying cells show through. When a header is directly above the first cell in its section (or when a footer is directly below the last cell in its section), there are no underlying cells to show through. In our case, we know each section has a header and footer. Thus, there is no need for fetching adjacent section/row data until at least the next section's header comes into view.

Let's switch Section 1, rows 3 and 4. With the debugger console in view, tap and hold index path {1, 4}, just over the three bars (signifying a movable row). Drag it upward until it switches places with index path {1, 3} but don't let go just. Observe the log:

```
tableView:targetIndexPathForMoveFromRowAtIndexPath:toProposedIndexPath:{1, 3}
```

Had we been logging this one more completely, it would have looked like this:

```
tableView:targetIndexPathForMoveFromRowAtIndexPath:{1, 4}toProposedIndexPath:{1
```

That's all the table view needs to do. Now let go, and the move is complete:

```
tableView:moveRowAtIndexPath:toIndexPath:{1, 3}
```

Again, a more complete log would have shown:

```
tableView:moveRowAtIndexPath:{1, 4}toIndexPath:{1, 3}
```

Tap Done in the navigation bar, and we see this:

```
tableView:canEditRowAtIndexPath:{1, 1}
tableView:canEditRowAtIndexPath:{1, 2}
tableView:canEditRowAtIndexPath:{1, 3}
tableView:canEditRowAtIndexPath:{1, 4}
tableView:canEditRowAtIndexPath:{2, 0}
tableView:canEditRowAtIndexPath:{2, 1}
tableView:canEditRowAtIndexPath:{2, 2}
tableView:canEditRowAtIndexPath:{2, 3}
tableView:canEditRowAtIndexPath:{2, 4}
```

Terrific! Now what other mischief can we cause? Let's delete section 2, row 1 (index path {2, 1}). Add some whitespace to the console, then tap edit once again, then the red circle at the left of the cell in section 2, row 1.

```
tableView:titleForDeleteConfirmationButtonForRowAtIndexPath:{2, 1}
```

Makes sense. The delete button appears (properly labeled). Now tap it to delete the row.

```
tableView:commitEditingStyle:forRowAtIndexPath:{2, 1}
```

First, we commit the editing style for index path {2, 1}. Then we effectively have a `reloadData` operation, which causes everything to be rechecked once again - headers, footers, row heights, the works. Only now we start at index path {0, 0} and run all the way clear through {24, 4}. So section 24 gets to go last this time!

Aaaand ... the app crashes big-time.

```
 *** Assertion failure in -[UITableView _endCellAnimationsWithContext:], /SourceC
1261.5/UITableView.m:920
 *** Terminating app due to uncaught exception 'NSInternalInconsistencyException'
```

Wha happen? Well, our faux data model still thinks there are 24 sections of five rows each. Obviously that doesn't match up with the table view's reality. In short, "table view fall down go boom."

Well, it was fun while it lasted :) Get the whole source code on Github[7] and we hope it'll be useful to you!

[7] https://github.com/akosma/DelegateOrder

MoMA and Software as an Art

Adrian Kosmaczewski

2012-02-01

What would be the place, in a museum like MoMA[1], of a collection of art dedicated to software?

If there is something that MoMA can make, is to boost your imagination. Anything is possible; the myriad of options for the expression of human creativity has no end, the mind boggles.

My dream has been, for years, to explain software, its intricacies, to make this part of our world accessible to anyone. Software rules our world, it is one of the most complex creations of man, yet it remains understood (albeit partly) by just a few.

There are many dimensions to software; the first to explore is size. When you tell anyone outside of the field that Windows 2000 took 5 years to a team of 1400 developers to complete[2], and that the whole thing is about 29 millions lines of code, it is still not enough; however, if you printed the whole code of Windows and put it in a series of books, how many books would it be?

On Kawara[3] has created a piece called “One Million Years”, on display at MoMA[4]; the whole thing is a series of books where the pages show, one after the other, as the name implies, one million years.

At 80 lines per page, at 1000 pages per volume, the source code of Windows 2000 would take… 363 volumes. Given that the Encyclopí¦dia Universalis or the Encyclopí¦dia Britannica consist of 20 or 30 volumes each, we are talking that a single company has been able to pull 12 encyclopí¦dias out of the hat for a single version of a product. I’m not talking about quality or other characteristics; just size, raw and simple.

That’s the magnitude of software. Now we can begin to understand the magnitudes, the cost, the implications.

Another magnitudes worth exploring would be cost, number of people involved, number of errors… Infographies would explain in detail the interconnections and the different dimensions, their relations, their

[1] http://www.moma.org/

[2] http://www.usenix.org/events/usenix-win2000/invitedtalks/lucovsky.ppt

[3] http://en.wikipedia.org/wiki/On_Kawara

[4] http://www.moma.org/collection/object.php?object_id=88213

impact. But again, the whole thing remains so virtual, so out of reach, so different of anything else, that we just run out of analogies in no time.

What other dimensions could be used?

Mobile Web Training in Zürich Once Again!

Adrian Kosmaczewski

2012-02-01

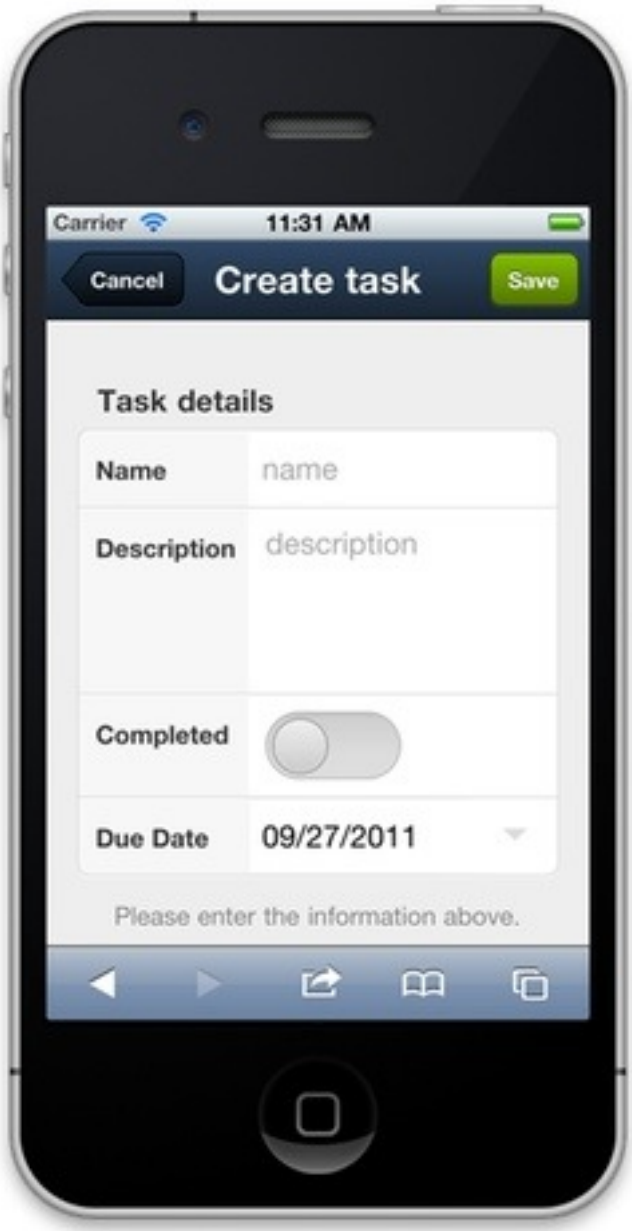
Simplificator and akosma software are thrilled to announce the second edition of our successful three day training about mobile web app development using jQuery Mobile, Sencha Touch and PhoneGap[1] in Zürich on March 14th, 15th and 16th!

Today, having a mobile application online is a must. But there are multiple platforms to write for, each with their own language, idioms and pitfalls. Luckily there is a simple solution that allows to write once and deploy on all modern mobile devices: HTML5 and JavaScript.

What others have been saying

- “Die Inhalte wurden lebendig und Step-by-Step präsentiert”
- “Merci, das war ein sehr lehrreicher Kurs”

[1] http://mobile-training.ch/



Easy, quick dev of “native” applications

This three day intensive course takes you from being a web developer to being a mobile developer. We take you through the basics of writing HTML5 applications for mobile devices, cover the additional APIs that allow you to access the functions of the devices (like storage, geo-location, accelerometers) and put you in control of deploying an application to either iOS or Android devices.

What

- Overview over what HTML5 brings for mobile development
- Overview over options for bringing web apps to a mobile device
- In depth review of jQuery Mobile and Sencha Touch
- JavaScript best practices
- JavaScript libraries that help building applications (Backbone.js, Raphaël.js)
- Bundling your app for the device with PhoneGap
- Accessing your devices sensors and special features from JavaScript

There will be lots of hands-on working building an application from scratch and bringing it to life on your mobile device. We will build two applications – one with jQuery Mobile, one with Sencha Touch and bring them to "life" as native applications on a mobile device.

Prerequisites

- HTML / CSS
- Basic JavaScript skills
- Programming experience
- Laptop with either iOS SDK (Mac OSX with XCode) or Android SDK (Mac, Windows, Linux) installed and running
- Mobile device (iOS, Android)

Class Size

Learning and working is best done in a relative small group. We plan on having maximum 9 people in this class so that we can spend enough time with each of our students.

Who

Jens-Christian Fischer, Simplificator GmbH

Jens-Christian started writing software in the late 1980s and been working on Web applications since the mid 1990s. The last 6 years he has been developing, writing and teaching Ruby on Rails and other web related technologies. Jens-Christian is TechLead and responsible for training at Simplificator GmbH[2], a Zurich based web development agency.

Adrian Kosmaczewski, akosma software

Adrian has been writing software for the past 20 years. He started working professionally in 1996, riding the first and second waves of the web. He started writing Cocoa applications for the Mac in 2002, and has been writing iOS apps since he returned from WWDC 2008.

[2] http://simplificator.com/

Adrian is the founder of akosma software, with a strong focus in all things iOS.

Course Details

Location

The 3 day course is held in Zurich at the offices of Simplificator GmbH, Pfingstweidstrasse 6, 8005 Zürich.

Price

- Regular Price: CHF 2100.– (inkl. 8% VAT)
- Early Bird Price: CHF 1785.– (inkl. 8% VAT) (a 15% discount). (Early bird price ends on 20.February 2012)
- Multiple people from the same company? Get a 10% discount for the second person.
- Members of /ch/open receive a 15% discount (not cumulative with early bird price)

Included in price

- 3 days of intensive hands-on training in a small group, with plenty of time to talk to the instructors
- Comprehensive Documentation
- Full lunch meal
- "There has to be food" – plenty of snacks and drinks during the day

Sign up

Use this form[3] to sign up! (Hurry up, in our first edition the training sold out after only 1 week!)

More Information

Don't hesitate to check <http:></http:> for updates, or to contact Simplificator[4] or to contact us[5] for more information.

Check out also the events in Facebook and LinkedIn (but remember that you have to sign up using the form in the site![6])

- Facebook event[7]

[3] https://docs.google.com/spreadsheet/viewform?formkey=dHhlLXRpWXVrWGcyNjRXV1dvSWRZaWc6MA

[4] mailto:kurs@simplificator.com

[5] /about/

[6] https://docs.google.com/spreadsheet/viewform?formkey=dHhlLXRpWXVrWGcyNjRXV1dvSWRZaWc6MA

[7] https://www.facebook.com/events/273212606079153/

- LinkedIn event[8]
- techup.ch[9]

[8] http://www.linkedin.com/osview/canvas?_ch_page_id=2&_ch_panel_id=3&_ch_app_id=30&_applicationId=2000&appParams=%7B%22event%22%3A916020%2C%22page%22%3A%22event%22%7D&_ownerId=0&completeUrlHash=mEVh

[9] http://techup.ch/550/mobile-web-training-in-zurich-once-again

QCon London 2012: Cross-Platform Mobile Track Announcement!

Adrian Kosmaczewski

2012-02-01

The Cross-Platform Mobile Track[1] of QCon London 2012[2] is ready to be announced! This year we'll have great speakers talking about how to create mobile applications using HTML5, JavaScript and CSS3:

[3]

- Maximiliano Firtman[4], the worldwide expert on mobile web application development, will talk about Mobile, HTML5 and the cross-platform promise[5].
- Jérome Giraud[6], creator of the Wink Toolkit[7] will present his creation in the talk Wink and the mobile web innovation[8].
- Andrea Giammarchi[9] from Nokia[10] will talk about Location aware mobile web apps with HTML5 and JavaScript[11].
- Tobie Langel[12] from Facebook[13] will give a yet undisclosed talk about an exciting new technology! Stay tuned for updates!
- Finally, Christophe Coenraets[14] from Adobe[15] will talk about

[1] http://qconlondon.com/london-2012/tracks/show_track.jsp?trackOID=569
[2] http://qconlondon.com/
[3] http://qconlondon.com/
[4] http://qconlondon.com/london-2012/speaker/Maximiliano+Firtman
[5] http://qconlondon.com/london-2012/presentation/Mobile,%20HTML5%20and%20the%20cross-platform%20promise
[6] http://qconlondon.com/london-2012/speaker/Jerome+Giraud
[7] http://www.winktoolkit.org/
[8] http://qconlondon.com/london-2012/presentation/Wink%20and%20the%20mobile%20web%20innovation
[9] http://qconlondon.com/london-2012/speaker/Andrea+Giammarchi
[10] http://www.developer.nokia.com/
[11] http://qconlondon.com/london-2012/presentation/Location%20aware%20mobile%20web%20app%20with%20HTML5%20and%20JavaScript
[12] http://qconlondon.com/london-2012/speaker/Tobie+Langel
[13] https://developers.facebook.com/
[14] http://qconlondon.com/london-2012/speaker/Christophe+Coenraets
[15] http://www.adobe.com/devnet.html

Cross-Platform Mobile Apps with HTML, JavaScript and PhoneGap[16].

All in all, an epic track with industry leaders speaking about a hot subject! This will happen on the Fleming room of the Queen Elizabeth II Conference Centre on Wednesday, March 7th.

We hope you see you in London in March!

TRACK	CROSS PLATFORM MOBILE
ROOM	Fleming
Day	Wednesday (7th Mar.)
10:30	Mobile, HTML5 and the cross-platform promise Maximiliano Firtman
11:50	Wink and the mobile web innovation Jerome Giraud
13:50	Location aware mobile web app with HTML5 and JavaScript Andrea Giammarchi
15:20	Cross Platform Mobile IV Tobie Langel
16:40	Cross-Platform Mobile Apps with HTML, JavaScript and PhoneGap Christophe Coenraets

[17]

[16] http://qconlondon.com/london-2012/presentation/Cross-Platform%20Mobile%20Apps%20with%20HTML,%20JavaScript%20and%20PhoneGap

[17] http://qconlondon.com/london-2012/tracks/show_track.jsp?trackOID=569

GOTO Copenhagen 2012: Call for Speakers!

Adrian Kosmaczewski

2012-02-06

I am very happy to announce that I will be the host of the Mobile Technologies: Native + Web[1] track of GOTO Copenhagen 2012[2]! The track will feature talks by Brian LeRoux, Graham Lee and Jérôme Giraud.

[3]

GOTO is a series of international software development conferences organized by Trifork[4] in Copenhagen, Amsterdam, Århus (Denmark) and Prague. This year the event will happen in Copenhagen on May 21st-25th.

We are currently looking for 2 more speakers for the conference; if you are interested, just contact us![5] We look forward to hearing from you.

[1] http://gotocon.com/cph-2012/tracks/show_track.jsp?trackOID=541
[2] http://gotocon.com/cph-2012/
[3] http://gotocon.com/cph-2012/
[4] http://www.trifork.com/
[5] /about/

Best Books of 2011

Adrian Kosmaczewski

2012-02-06

Just like in 2010[1], 2009[2], 2008[3] and 2007[4], here goes the traditional book of the year post for 2011! This year my reading list included design, history, and lots of JavaScript.

Here goes the list, in a completely arbitrary order of personal preference:

"Design for Hackers"[5] by David Kadavy

It all started, around August, with a conversation with Paul, my cousin from London, the founder of Zerofee[6]; we were talking that while designers could easily find tutorials and documentation to learn about software development, it was much harder for developers to find material about design, at least to learn the basic concepts.

Somehow, David Kadavy must have heard us, and he came up the following month with this great book. I have recommended it to every one of my students since I read it; it is built like a computer book, with clear explanations, diagrams and lots of pictures; every concept is described in detail, including historical references and examples.

David provides a healthy introduction to design, going through subjects such as fonts, proportions, color theory, structure, grids, and much, much more.

I insist; this is a must read for any developer, particularly those who, like me, are self-taught and eager to expand their own possibilities.

"Steve Jobs"[7], by Walter Isaacson

The day Steve Jobs passed away I was in South Africa, giving some trainings in Johannesburg, precisely about iOS. I remember waking up,

[1] /blog/best-books-of-2010/
[2] /blog/best-books-of-2009/
[3] /blog/best-books-of-2008/
[4] /blog/best-books-of-2007/
[5] http://www.designforhackers.com/
[6] http://www.zerofee.org/
[7] http://www.amazon.com/Steve-Jobs-Walter-Isaacson/dp/1451648537

opening my copy of Echofon in my iPhone, and seeing lots of tweets with just an apple sign on them.

I said to myself, as I started to scroll downwards, that this was it. I sat on the edge of the bed, realizing that it was the end of a huge chapter for the computer industry.

That very same morning, I gave an introduction to iOS to some developers, and of course there was a special thing to that training. Somehow there was a legacy of the guy in every NSObject that we allocated in memory.

Later that same month, I went to the USA and landed in Newark. While waiting for my connecting flight, I was dragging my feet in the terminal, and the corner of my eye saw the book in the shelves of a bookstore.

I read it in about 5 days. There had not been a book that hook me as much as this one in ages.

I will not go as far as saying that this was the book of the year. I won't add anything to what is already available online about it. It was, without any doubt, the most hyped book of the decade. And it is a surprisingly good one; not biased, very acid in some parts - I guess Jobs would not have liked reading some sections of it. I picture him throwing the book out of the window, outraged, while weeping at some chapters.

I admit, I've shed some tears in some parts. All in all, a very emotional piece, not to be missed.

"JavaScript Patterns"[8], by Stoyan Stefanov

2011 was the year of the mobile web. Not only because some analyst said so, but because the demand for mobile web solutions from companies has increased dramatically in one year. Also because the capabilities of smartphones have grown in such a way that today, web apps are a viable choice for consumers.

This means, by all standards, that JavaScript regains a preeminence on the web; longtime a language that was bashed and forgotten, JavaScript reappears in front of many developers as the instrument by which the mobile web becomes a reality.

This book is a perfect way to rediscover JavaScript; to forget the pain of the past, to see that it is a wonderful language that, as Crockford would say, was hugely misunderstood.

By the way, readers should have read Crockford's "JavaScript: the Good Parts" before this book; Stefanov builds on top of that knowledge and provides the developer with a fresh bouquet of idioms and constructions that will be useful in every JavaScript project.

[8] http://shop.oreilly.com/product/9780596806767.do

Finally, given the rise of Node.js in the past few years, reading it provides also with a solid background for the next wave of JavaScript frameworks hitting the market these days.

"Programming the Mobile Web"[9], by Maximiliano Firtman

Maximiliano is a genius. The guy has pulled the complete bible, the absolute reference, for everything that has to do with mobile web development. The book is a treasure of capabilities, comparisons, history, and nitty-gritty details about every possible mobile web browser in the planet.

How he does it, it's his great mystery. After publishing this book, he came up with the great Mobile HTML5 site[10] which, if you haven't seen it yet, you should.

Even better, he's argie like I am, and I've had the opportunity of inviting him to Zürich last year, to hear him talk about jQuery Mobile. Which reminds me of…

"jQuery Mobile: Up and Running"[11], again by Maximiliano Firtman

… his latest book; this time, Maximiliano tackled the hottest mobile framework of the moment. I've read the book in "early release" mode, prior to the final publication, and so far it looks very promising.

In this book, Maximiliano explains the core concepts of jQuery Mobile, the semantics and the capabilities of the framework, clearly explaining its strengths as well as its weaknesses.

I have learnt a lot about jQuery Mobile through this book, so I strongly recommend it to anyone interested in the subject.

"Mobile Design for iPhone and iPad"[12], by Smashing Magazine

A very nice and concise eBook by the great people of Smashing Magazine, with great tips and tricks about how to create UIs for the new generation of touchscreen devices. I've learnt a lot with this book.

[9] http://shop.oreilly.com/product/9780596807795.do

[10] http://mobilehtml5.org/

[11] http://shop.oreilly.com/product/0636920014607.do

[12] http://www.smashingmagazine.com/2010/11/03/ebook-4-mobile-design-for-iphone-and-ipad/

“iPad at Work”[13], by Apple

Finally, a nice free eBook by Apple, very useful for explaining the iPad and its multiple capabilities to business people; I am personally seeing more and more iPads in enterprise contexts, so I think that this is an important (and small) title to read.

[13] http://itunes.apple.com/us/book/ipad-at-work/id455380218?mt=11

Swiss App Awards Nominees!

Adrian Kosmaczewski

2012-02-24

The jury of the Swiss App Awards[1] has just published the official list of nominees in each category!

Best Game App

- Herbert the Misanthropical Fly, Etite
- Memory pour enfants Bimbadaboum, Atipik Creative Factory
- MonsterUp, Kariosgames.com
- Pilotifant, Millform AG
- Pingwin Adventures, FEINHEIT GmbH

Best User Experience App

- Koubachi, Koubachi AG
- Memory pour enfants Bimbadaboum, Atipik Creative Factory
- Pilotifant, Millform AG
- SBB Mobile, SBB AG
- Stations – Swiss Public Transport , André Horstmann

Best Bank App

- BudgetBook, noidentity gmbh
- MoneyBook, noidentity gmbh
- Swissquote, Swissquote
- UBS Mobile Banking, UBS AG
- UBS KeyClub, UBS AG

Best Web App

- AppAware – A Social Network for App Discovery, 42matters AG
- Kooaba Déjà Vu, kooaba AG
- Mobility Car, Mobility Cooperative
- QUENTIQ Tracker, Quentiq
- Spocal, Spocal

[1] http://swissappawards.ch/

Most Downloaded App

- 20 Minuten Online .ch, 20 Minuten
- local.ch, local.ch
- SBB mobile, SBB AG
- Swisscom Fan-Glocke, Swisscom AG
- TCS, Touring Club Schweiz

App of the Year

- Koubachi, Koubachi AG
- Memory pour enfants Bimbadaboum, Atipik Creative Factory
- MonsterUp, Kariosgames.com
- Pilotifant, Millform AG
- SBB mobile, SBB AG

Stay tuned for more information in the official Twitter account of the Swiss App Awards[2] as we approach the day of the ceremony, on March 21st!

[3]

[2] https://twitter.com/swissappawards
[3] http://swissappawards.ch/

This week: QCon London 2012!

Adrian Kosmaczewski

2012-03-05

This week, akosma software will be in London hosting the Cross-Platform Mobile Track[1] of QCon London 2012[2]! This year we have great speakers talking about how to create mobile applications using HTML5, JavaScript and CSS3:

[3]

- Maximiliano Firtman[4], the worldwide expert on mobile web application development, will talk about Mobile, HTML5 and the cross-platform promise[5].
- Jérome Giraud[6], creator of the Wink Toolkit[7] will present his creation in the talk Wink and the mobile web innovation[8].
- Andrea Giammarchi[9] from Nokia[10] will talk about Location aware mobile web apps with HTML5 and JavaScript[11].
- Tobie Langel[12] from Facebook[13] will give a yet undisclosed talk about an exciting new technology! Stay tuned for updates!
- Finally, Christophe Coenraets[14] from Adobe[15] will talk about Cross-Platform Mobile Apps with HTML, JavaScript and Phone-

[1] http://qconlondon.com/london-2012/tracks/show_track.jsp?trackOID=569
[2] http://qconlondon.com/
[3] http://qconlondon.com/
[4] http://qconlondon.com/london-2012/speaker/Maximiliano+Firtman
[5] http://qconlondon.com/london-2012/presentation/Mobile,%20HTML5%20and%20the%20cross-platform%20promise
[6] http://qconlondon.com/london-2012/speaker/Jerome+Giraud
[7] http://www.winktoolkit.org/
[8] http://qconlondon.com/london-2012/presentation/Wink%20and%20the%20mobile%20web%20innovation
[9] http://qconlondon.com/london-2012/speaker/Andrea+Giammarchi
[10] http://www.developer.nokia.com/
[11] http://qconlondon.com/london-2012/presentation/Location%20aware%20mobile%20web%20app%20with%20HTML5%20and%20JavaScript
[12] http://qconlondon.com/london-2012/speaker/Tobie+Langel
[13] https://developers.facebook.com/
[14] http://qconlondon.com/london-2012/speaker/Christophe+Coenraets
[15] http://www.adobe.com/devnet.html

Gap[16].

An awesome moment with industry leaders speaking about a hot subject! This will happen on the Fleming room of the Queen Elizabeth II Conference Centre next Wednesday. See you there!

TRACK	CROSS PLATFORM MOBILE
ROOM	Fleming
Day	Wednesday (7th Mar.)
10:30	Mobile, HTML5 and the cross-platform promise Maximiliano Firtman
11:50	Wink and the mobile web innovation Jerome Giraud
13:50	Location aware mobile web app with HTML5 and JavaScript Andrea Giammarchi
15:20	Cross Platform Mobile IV Tobie Langel
16:40	Cross-Platform Mobile Apps with HTML, JavaScript and PhoneGap Christophe Coenraets

[17]

[16] http://qconlondon.com/london-2012/presentation/Cross-Platform%20Mobile%20Apps%20with%20HTML,%20JavaScript%20and%20PhoneGap

[17] http://qconlondon.com/london-2012/tracks/show_track.jsp?trackOID=569

Mobile Web Training in Zürich!

Adrian Kosmaczewski

2012-03-13

This week we will be in Zürich, giving the second edition of our three day long mobile web training[1] in partnership with Simplificator[2]!

[3]

We are going to teach developers how to create mobile web applications using Sencha Touch[4], PhoneGap[5] and jQuery Mobile[6].

Stay tuned for more dates in the future, and also new locations!

[1] http://mobile-training.ch/
[2] http://simplificator.com/
[3] http://www.sencha.com/products/touch/
[4] http://www.sencha.com/products/touch/
[5] http://phonegap.com/
[6] http://jquerymobile.com/

Tonight is Swiss App Awards 2012 Night!

Adrian Kosmaczewski

2012-03-21

We are thrilled to remember you that tonight will be held the ceremony of the first Swiss App Awards[1]! It will be held from 18:30 at The Millennium Room of the Hotel Marriott (Neumühlequai 42, Zürich). We are going to celebrate the incredible quality and work that the Swiss are injecting into mobile app stores!

Adrian, member of the jury, will be posting news, photos and videos on our Twitter account[2] and on our Facebook page[3]. You can also follow the official Twitter account[4] of the Swiss App Awards for more information.

Just as a reminder, here are the nominees in each category:

Best Game App

- Herbert the Misanthropical Fly, Etite
- Memory pour enfants Bimbadaboum, Atipik Creative Factory
- MonsterUp, Kariosgames.com
- Pilotifant, Millform AG
- Pingwin Adventures, FEINHEIT GmbH

Best User Experience App

- Koubachi, Koubachi AG
- Memory pour enfants Bimbadaboum, Atipik Creative Factory
- Pilotifant, Millform AG
- SBB Mobile, SBB AG
- Stations – Swiss Public Transport , André Horstmann

Best Bank App

- BudgetBook, noidentity gmbh

[1] http://swissappawards.ch/
[2] https://twitter.com/akosmasoftware
[3] https://www.facebook.com/akosmasoftware
[4] http://twitter.com/SwissAppAwards

- MoneyBook, noidentity gmbh
- Swissquote, Swissquote
- UBS Mobile Banking, UBS AG
- UBS KeyClub, UBS AG

Best Web App

- AppAware – A Social Network for App Discovery, 42matters AG
- Kooaba Déjà Vu, kooaba AG
- Mobility Car, Mobility Cooperative
- QUENTIQ Tracker, Quentiq
- Spocal, Spocal

Most Downloaded App

- 20 Minuten Online .ch, 20 Minuten
- local.ch, local.ch
- SBB mobile, SBB AG
- Swisscom Fan-Glocke, Swisscom AG
- TCS, Touring Club Schweiz

App of the Year

- Koubachi, Koubachi AG
- Memory pour enfants Bimbadaboum, Atipik Creative Factory
- MonsterUp, Kariosgames.com
- Pilotifant, Millform AG
- SBB mobile, SBB AG

[5]

[5] http://swissappawards.ch/

Swiss App Awards 2012 Winners!

Adrian Kosmaczewski

2012-03-22

We proudly present the winners of Swiss App Awards 2012[1]:

- Best User Experience App: Pilotifant[2] for iOS, by WIRZ/Millform AG
- Most Downloaded App: SBB Mobile[3] for multiple platforms, by SBB AG
- Best Web App: AppAware[4] for Android, by 42matters AG
- Best Game App: MonsterUp[5] for Windows Phone, by karios-games.com
- Best Bank App: MoneyBook[6] for iOS, by noidentity gmbh
- People's Prize: Swisscom Fan-Glocke[7] for iOS, by Swisscom AG/DU DA Group/Saatchi&Saatchi
- App of the Year: Pilotifant[8] for iOS, by WIRZ/Millform AG

Interestingly enough, the contest was truly cross-platform, and there were winners on iOS, Android, Windows Phone and other platforms.

Congratulations to all, and thanks to everyone who contributed apps and/or came to the ceremony! You can check more pics of the ceremony in the album in our Facebook page[9]. Check it out!

[1] http://swissappawards.ch/

[2] http://itunes.apple.com/us/app/pilotifant/id443757518?mt=8

[3] http://www.sbb.ch/en/timetable/mobile-timetables/mobile-apps.html

[4] https://play.google.com/store/apps/details?id=com.appaware

[5] http://www.windowsphone.com/en-US/apps/e51733bd-cd13-e011-9264-00237de2db9e

[6] http://moneybookapp.com/moneybook.html

[7] http://itunes.apple.com/ch/app/fan-glocke/id488729620?mt=8

[8] http://itunes.apple.com/us/app/pilotifant/id443757518?mt=8

[9] https://www.facebook.com/media/set/?set=a.10150693538661100.420407.184046196099&type=1

Tonight in Geneva: Introduction to Sencha Touch 2

Adrian Kosmaczewski

2012-03-28

Tonight Adrian will be presenting Sencha Touch 2[1] to a select group of JavaScript fanatics!

This presentation will happen in the University of Geneva, in the "Sciences III" building (thanks @yannis_[2] for the room and @bdufresne[3] for the organization!), Boulevard d'Yvoy near the Jonction neighborhood, at 7pm tonight. Please RSVP in the official Meetup site[4] of the JavaScript Genève group (@jsgeneve[5] in Twitter).

See you there!

[1] http://www.meetup.com/jsgeneve/events/54491632/
[2] https://twitter.com/yannis_
[3] https://twitter.com/bdufresne
[4] http://www.meetup.com/jsgeneve/events/54491632/
[5] https://twitter.com/jsgeneve

Size Matters

Adrian Kosmaczewski

2012-03-29

One of the facts I vividly remember of studying physics in university (this was in the mid 90's in Geneva, Switzerland) was a certain disconnection between Relativity and Quantum Mechanics. The former is a theory used to describe phenomena at macro level, like galaxies, planets, stellar systems, while the latter describes the interactions at micro level, the atoms, light, particles, energy at microscopic levels.

When you apply some relativity equations to atoms, you get results that are not supported by experimentation; and the same happens when you apply some quantum theory equations to objects like planets. It is not that all the relativistic facts do not apply at micro level, or that all quantum facts do not apply in macro level; it is that there is no unifying theory that explains everything, and this quest is the graal of modern physics.

Fast-forward 5 years.

One of the facts I vividly remember of studying Economics in university (this time in Buenos Aires) was a certain disconnection between Microeconomy and Macroeconomy. The models that describe the behavior of the consumer (which is the heart of the study of Microeconomy) yield wrong conclusions when applied to issues like unemployment, foreign trade or other matters that are usually better explained by Macroeconomy; and similarly, well, you get the picture.

As a matter of fact, my Macroeconomics teacher would say that whatever we learnt in Microeconomics class was wrong, and that he had the right answers; of course, the Micro teacher told us the same the year before.

Fast-forward 5 years.

One of the facts that I vividly remember of studying Computer Science during my master degree program was a certain disconnection between small and big software projects. What works in small, simple applications and systems, including human and technical factors, does not usually work in bigger, more complex projects.

It is not the same to work on a startup project with some friends in a garage to create the next social networking site, where coordination is easy, most of the tools required are available for free, where

the projects rarely have any dedicated quality assurance team, than working in, say, a bigger organization like Microsoft, together with other 1500 engineers and testers, all dedicated full time to writing and testing the next version of Windows.

The hardware requirements are not the same, either; in small projects you could use a couple of Mac Minis and a cloud hosting service and you are done; at Google they have MapReduce and a couple hundred thousand computers in a datacenter with air conditioning and security 24/7, and they still require more infrastructure every day.

However, and this is my main point, there is no proven recipe that can help a company grow from 10 to 10'000 people and from 10 to 10 million customers in a snap; there are some good techniques and principles, here and there, to make your software grow; but nobody actually knows of a generic recipe for every software company.

There are so many factors in macro problems, that the interaction of those factors has to be taken into account; not only the factors themselves, but also their interdependencies. I guess you see where I am going with this. This problem is usually called scaling in computer circles, and I think that the word can be applied to economy and physics.

As humans, we have trouble scaling. Scale is important in our eyes, because we tend to think that bigger is better. Bigger is more money, in general, but not necessarily better; we have trouble going from small to big and vice versa. Not only in facts; also in concepts. We cannot foresee the implications of scaling. At least, not completely, and not so far.

This disconnection creates lots of problems in our society. Politicians forget the human being altogether, buried beneath tons of numbers and statistics. Voters do not understand that managing a country is not like managing your household economy. Schools do not teach how to solve scalability problems; heck, they do not even properly teach kids how to work in teams to solve small, micro problems.

Small companies do not understand that scaling is neither automatic nor a required process, and that not all companies should grow; some companies work better when small than when they grow up, and that's why they sometimes fail. Venture capitalists that are not familiar with technology cannot understand this fact, and will sometimes sacrifice good working teams for just making more money or for getting into the stock market.

The knowledge we have about the problem of scaling is limited; I actually sometimes ask myself whether there is a solution to it, that would justify the search of a global theory in physics, a unified theory in economics, or a generic scaling procedure for companies and software systems.

I do not have the answer; the fact is that size matters, and that this pattern has to do with the world we are living in; it does not matter

whether you are a physicist, an economist or a programmer; this is how the world works.

Vidéo de la Présentation de Sencha Touch 2

Adrian Kosmaczewski

2012-04-01

Voici la vidéo de la présentation de Adrian à propos de Sencha Touch 2, lors de la réunion du groupe JavaScript Genève le 28 mars dernier.

Un grand merci à notre grand ami Bertrand Dufresne pour la production et la “mise en boîte” de cette vidéo! Ne loupez pas les prochains meetings JavaScript Genève[1]!

[1] http://www.meetup.com/jsgeneve/

Formation de Développement d'Applications Web Mobiles à Genève

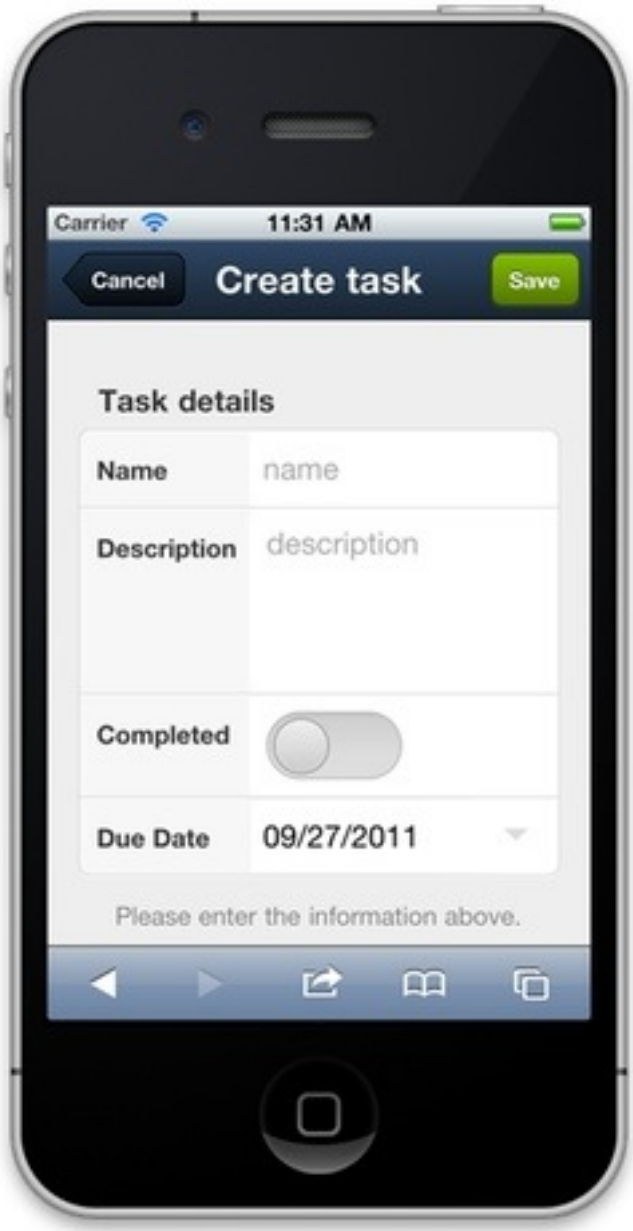
Adrian Kosmaczewski

2012-04-16

Après le succès des éditions de Zürich[1] et Afrique du Sud[2], nous sommes ravis de présenter la première édition du cours de développement d'applications web mobiles à Genève, en français![3] Le cours est organisé avec la précieuse collaboration de Bertrand Dufresne, organisateur du groupe JS Genève[4].

Inscrivez-vous dès maintenant[5] et profitez d'une réduction "Early Bird" de 15% jusqu'au 25 avril prochain.

[1] http://mobile-training.ch/
[2] http://www.appdev.co.za/
[3] http://apprendrewebmobile.com/
[4] http://www.meetup.com/jsgeneve/
[5] https://apprendre.wufoo.com/forms/formation-applications-web-mobiles/



Notre cours intensif de 3 jours s’adresse aux développeurs web qui souhaitent utiliser des technologies qu’ils connaissent déjà pour développer des applications mobiles de qualité, compatible avec la majorité des périphériques mobiles disponibles sur du marché. Nous proposons d’accéder au statut de développeur d’application web mobiles.

La formation s'articule autour de jQuery Mobile[6], Sencha Touch[7] et PhoneGap[8] (Cordova[9]), qui sont, à notre avis, les plus importantes technologies disponibles à l'heure actuelle pour développer des applications web mobiles basées sur HTML5[10].

Vous apprendrez les bases nécessaires à l'écriture d'applications web professionnelles pour périphériques mobiles (smartphones ou tablettes) en HTML5 et en JavaScript. Nous vous accompagnerons dans la découvertes les APIs specifiques aux terminaux mobiles tactiles telles que le stockage local, la géo-localisation, l'accéléromètre, le gyroscope et la boussole. Nous vous expliquerons comment déployer vos applications web mobiles professionnelles sur iOS et Android.

Nombre de participants

Ce cours est organisé à partir de 6 participants et est limité à 12 personnes maximum, afin de permettre à nos deux formateurs de vous aider individuellement.

Matériel nécessaire

Vous devez apporter votre propre ordinateur portable. Bien entendu, tous les systèmes d'exploitation sont les bienvenus (OS X, Linux ou Windows).

Avant le cours et suivant votre système d'exploitation, vous devez avoir téléchargé et installé au préalable les éléments suivants:

- Pour OS X: Xcode 4.3.2 et/ou le Android SDK.
- Windows: Android SDK et/ou Visual Studio avec les outils de développement de Windows Phone.
- Linux: Android SDK.
- Nous vous demandons aussi d'installer aussi la dernière version de Google Chrome, ou de Safari pour les utilisateurs Mac.

Lieu

Le cours sera donné dans la salle de réunion de l'hôtel Eden Genève.

Eden Hôtel Genève
Rue de Lausanne 135
1202 Genève

[6] http://jquerymobile.com/
[7] http://www.sencha.com/products/touch/
[8] http://phonegap.com/
[9] http://incubator.apache.org/cordova/
[10] http://html5.org/

Coût

Le prix du cours est de CHF 2’100.00 (hors taxes).

Profitez des offres suivantes (non cumulables):

- Nous vous offrons une réduction de 15% (soit CHF 1’785.00 HT) pour toute inscription effectuée avant le 25 avril 2012.
- Si vous travaillez dans une même entreprise, une réduction de 10% est appliquée dès inscriptions d’un deuxième collaborateur.
- Les membres du usergroup JavaScript Genève[11] qui ont au moins participé à un Meetup disposent d’une réduction de 15% non limitée dans le temps.

Des questions?

Nous restons à votre disposition pour toute question que vous pourriez avoir en relation avec nos formations: cours@akosma.com[12] ou sur notre site.

[11] http://www.meetup.com/jsgeneve/
[12] mailto:cours@akosma.com

Third Edition of the Mobile Web Training in Zürich

Adrian Kosmaczewski

2012-04-16

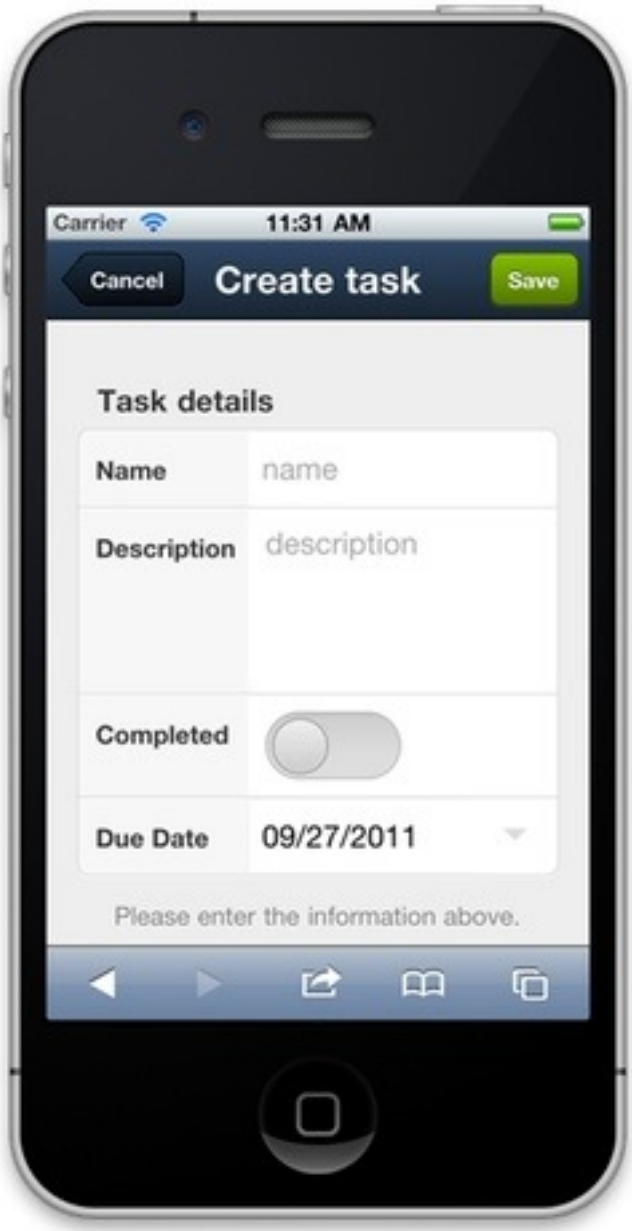
Simplificator[1] and akosma software are thrilled to announce the third edition of our successful three day training about mobile web app development using jQuery Mobile, Sencha Touch and PhoneGap[2] in Zürich on May 16th, 17th and 18th!

Today, having a mobile application online is a must. But there are multiple platforms to write for, each with their own language, idioms and pitfalls. Luckily there is a simple solution that allows to write once and deploy on all modern mobile devices: HTML5 and JavaScript.

What others have been saying

- “Die Inhalte wurden lebendig und Step-by-Step präsentiert”
- “Merci, das war ein sehr lehrreicher Kurs”

[1] http://simplificator.com/
[2] http://mobile-training.ch/



Easy, quick dev of “native” applications

This three day intensive course takes you from being a web developer to being a mobile developer. We take you through the basics of writing HTML5 applications for mobile devices, cover the additional APIs that allow you to access the functions of the devices (like storage, geo-location, accelerometers) and put you in control of deploying an application to either iOS or Android devices.

What

- Overview over what HTML5 brings for mobile development
- Overview over options for bringing web apps to a mobile device
- In depth review of jQuery Mobile and Sencha Touch
- JavaScript best practices
- Bundling your app for the device with PhoneGap
- Accessing your devices sensors and special features from JavaScript

There will be lots of hands-on working building an application from scratch and bringing it to life on your mobile device. We will build two applications – one with jQuery Mobile, one with Sencha Touch and bring them to “life” as native applications on a mobile device.

Prerequisites

- HTML / CSS
- Basic JavaScript skills
- Programming experience
- Laptop with either iOS SDK (Mac OSX with XCode) or Android SDK (Mac, Windows, Linux) installed and running
- Mobile device (iOS, Android)

Class Size

Learning and working is best done in a relative small group. We plan on having maximum 9 people in this class so that we can spend enough time with each of our students.

Who

Jens-Christian Fischer, Simplificator GmbH

Jens-Christian started writing software in the late 1980s and been working on Web applications since the mid 1990s. The last 6 years he has been developing, writing and teaching Ruby on Rails and other web related technologies. Jens-Christian is TechLead and responsible for training at Simplificator GmbH[3], a Zurich based web development agency.

Adrian Kosmaczewski, akosma software

Adrian has been writing software for the past 20 years. He started working professionally in 1996, riding the first and second waves of the web. He started writing Cocoa applications for the Mac in 2002, and has been writing iOS apps since he returned from WWDC 2008. Adrian is the founder of akosma software, with a strong focus in all things mobile.

[3] http://simplificator.com/

Course Details

Location

The 3 day course is held in Zurich at the offices of Simplificator GmbH, Pfingstweidstrasse 6, 8005 Zürich.

Price

- Regular Price: CHF 2100.– (inkl. 8% VAT)
- Early Bird Price: CHF 1785.– (inkl. 8% VAT) (a 15% discount). (Early bird price ends on 20.February 2012)
- Multiple people from the same company? Get a 10% discount for the second person.
- Members of /ch/open receive a 15% discount (not cumulative with early bird price)

Included in price

- 3 days of intensive hands-on training in a small group, with plenty of time to talk to the instructors
- Comprehensive Documentation
- Full lunch meal
- “There has to be food” – plenty of snacks and drinks during the day

Sign up

Use this form[4] to sign up! (Hurry up, our two first editions sold out after only 1 week!)

More Information

Don’t hesitate to check <http:></http:> for updates, or to contact Simplificator[5] or to contact us[6] for more information.

[4] https://docs.google.com/spreadsheet/viewform?formkey=dGhNazZrdzF1Q3llVjdMS3p5NkZleUE6MA#gid=0

[5] mailto:kurs@simplificator.com

[6] /about/

Getting the Next and the Previous NSIndexPath Instances

Adrian Kosmaczewski

2012-04-20

Very often, when you work with `UITableViewControllers` driven by `NSFetchedResultsControllers`, that you want to get the "previous" or the "next" elements in the results controller. Visually, this operation corresponds, from the point of view of the user, to select the cell that sits immediately above or below from the currently selected one.

Of course, you can't just ++ on the current `NSIndexPath`, because these objects have both a `section` and a `row` component, and the math required to jump from one to the other can be quite cumbersome; instead, it would be useful to have a reusable set of methods, and avoid the clutter in our controllers.

I have created a very simple category on the `NSFetchedResultsController` class that retrieves the "next" and the "last" `NSIndexPath` given any other `NSIndexPath`; this can be dropped and reused in your projects and is very simple to use.

Interface

```
#import "CoreData/CoreData.h"

@interface NSFetchedResultsController (AKOLibrary)

- (NSIndexPath *)ako_incrementIndexPath:(NSIndexPath *)oldIndexPath;

- (NSIndexPath *)ako_decrementIndexPath:(NSIndexPath *)oldIndexPath;

@end
```

Implementation

```
#import "NSFetchedResultsController+AKOLibrary.h"

@implementation NSFetchedResultsController (AKOLibrary)

- (NSIndexPath *)ako_incrementIndexPath:(NSIndexPath *)oldIndexPath
```

```
{
    NSIndexPath *nextIndexPath = nil;

    id <NSFetchedResultsSectionInfo> sectionInfo = [[self sections] objectAtInd
    NSInteger rowCount = [sectionInfo numberOfObjects];

    NSInteger nextRow = oldIndexPath.row + 1;
    NSInteger currentSection = oldIndexPath.section;
    if (nextRow = 0)
    {
        previousIndexPath = [NSIndexPath indexPathForRow:nextRow inSection:curr
    }
    else
    {
        NSInteger nextSection = currentSection – 1;
        if (nextSection &gt;= 0)
        {
            previousIndexPath = [NSIndexPath indexPathForRow:0 inSection:nextSe
        }
    }

    return previousIndexPath;
}

@end
```

How to use

To use this class, just do the following:

```
NSIndexPath *indexPath = [NSIndexPath indexPathForRow:19 inSection:4];

NSIndexPath *next = [controller ako_incrementIndexPath:indexPath];
NSIndexPath *previous = [controller ako_decrementIndexPath:indexPath];
```

If the `NSIndexPath` (19,4) was the last of whole table, `next` would be `nil`. The same, if `NSIndexPath` was the first (0, 0), then `previous` would be `nil`. In all other situations, the `next` and `previous` index paths will correspond to the cells immediately above or below from the current table.

Hope this helps!

Introducing the Teaching Editor

Adrian Kosmaczewski

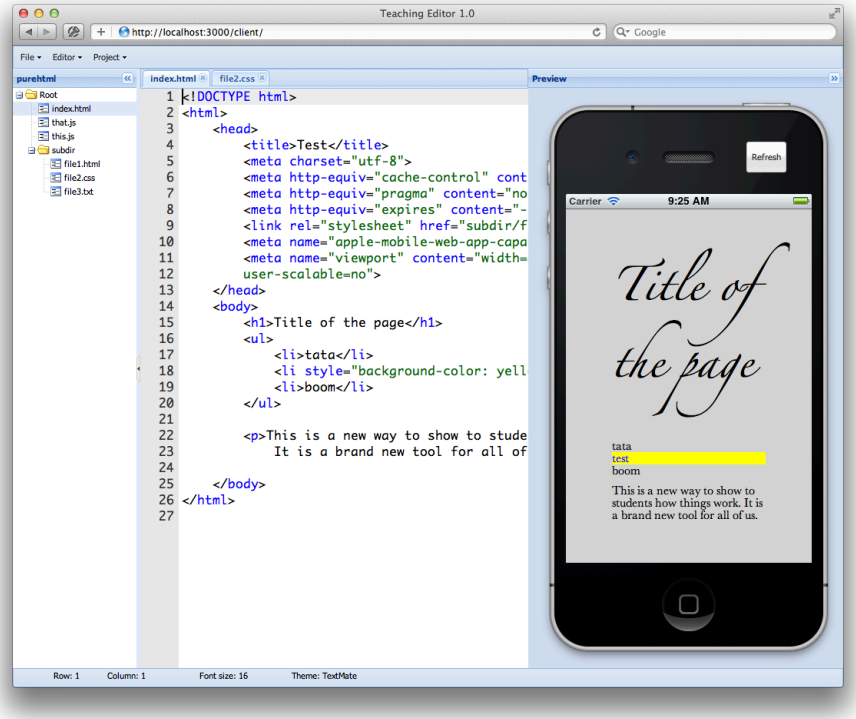
2012-04-23

We are very happy to introduce our latest open source project: the Teaching Editor[1]. This project provides an online editor that automatically reloads the contents of an iPhone-sized frame. It also provides students with a read-only mode, allowing them to follow in real time whatever code is written in the screen of the teacher, and they can also download the current state of the code at any moment.

It is built exclusively in JavaScript, using the following libraries:

- Ace[2]
- Ext.js[3]
- Node.js[4]
- Express[5]

[1] https://akosma.github.io/TeachingEditor/
[2] http://ace.ajax.org/
[3] http://www.sencha.com/products/extjs/
[4] http://nodejs.org/
[5] http://expressjs.com/



Requirements

Server

Use Homebrew[6] to install Node.js[7] in your system. Also, install npm[8] and install Express[9] and Socket.IO[10] with npm.

The `install.sh` script performs all the required operations to install external dependencies in your system.

Client

The client has been tested successfully on several combinations of operating systems and browsers:

- Cross-platform browsers:
  - Firefox 10
  - Chrome 17
  - Opera 11
- OS X “Lion”

[6] http://mxcl.github.com/homebrew/
[7] http://nodejs.org/
[8] http://npmjs.org/
[9] http://expressjs.com/
[10] http://socket.io/

  - Safari 5.1
- Windows 7
  - Internet Explorer 9
- iOS
  - Mobile Safari for iOS 5.1 on the iPad (in this case, however, scrolling is not possible)

Pay attention to the fact that the mobile libraries themselves might not be compatible with some of these browsers (in particular, Sencha Touch only works on Webkit-based browsers).

How to Use

- `./install.sh` (this will download the required libraries, only required once)
- `./launch.js` (this launches the Node.js app and opens a browser window)
- Students can browse to the IP shown in the dialog of the "Project / Show Share URL" menu entry.

License

This project is released under the GPLv3 license. Please check the LICENSE[11] file for details.

[11] https://github.com/akosmasoftware/TeachingEditor/blob/master/LICENSE

Our Open Source Projects

Adrian Kosmaczewski

2012-04-24

A quick reminder of our most popular open source projects on Github:

- bluewoki[1]
- Cortito[2]
- CoreTextWrapper[3]
- dotfiles[4]
- eBook-Template[5]
- iPhoneWebServicesClient[6]
- nib2objc[7]
- Senbei[8]
- Teaching Editor[9]

Feel free to fork and enjoy!

[1] http://bluewoki.com/
[2] http://akosma.github.com/cortito/
[3] http://akosma.github.com/CoreTextWrapper/
[4] http://akosmasoftware.github.com/dotfiles/
[5] http://akosmasoftware.github.com/eBook-Template/
[6] http://akosma.github.com/iPhoneWebServicesClient/
[7] http://akosma.github.com/nib2objc/
[8] http://akosmasoftware.github.com/Senbei/
[9] http://akosmasoftware.github.com/TeachingEditor/

Mi Abuela Herta

Adrian Kosmaczewski

2012-04-29

Mi abuela materna se llamaba Herta Schlerff.

Murió cuando yo tenía 11 años. Encadenó una serie de problemas de salud crónicos, entre un par de paros cardíacos y una fractura de cadera, pero murió tranquilamente, en su sueño, una mañana de abril de 1985, en la Clínica Olivos, ahí en Arenales y Maipú.

La velamos en la casa de sepelios que estaba en la avenida Maipú, enfrente de la quinta presidencial. Luego cremamos sus restos y los esparcimos en el Río de la Plata.

Pero no sirve de nada que les cuente tantos detalles sobre su muerte, sin antes contarles más sobre ella y su vida.

La abuela Herta (como siempre la llamé, un poco para distinguirla de mi otra abuela, Janina) nació el 31 de diciembre de 1903 en Filipópolis, una ciudad de Bulgaria que luego de varias guerras mundiales pasó a llamarse Plovdiv. O fue al revés, no me acuerdo. La cosa es que la familia Schlerff, alemanes de culto protestante, eran floristas. En aquella época, aparentemente, eran los floristas más importantes de Europa Oriental: incluso fueron los proveedores oficiales, a fines del siglo XIX, del sultán de Turquía, lo cual supongo yo, en tiempos del Imperio Otomano, no era moco de pavo.

La abuela Herta tenía cinco hermanas, todas con nombres más raros unas que otras: Mitzi, Reemda, y otros nombres que me he olvidado en este momento. De las seis hermanas, tres se casaron y tres quedaron solteras, ocupándose de su madre, una tal Ana Havel, una señora de carácter que marcaba el paso de toda la familia.

Por razones que no me quedaron claras, la abuela Herta hizo la escuela primaria en Alejandría, en Egipto. Aparentemente uno de sus compañeritos de escuela era un tal Rudolf Hess, y cotejando edades llegué a la conclusión de que es el mismo Rudolf Hess del que hablan, lamentablemente y con razón, los libros de historia.

La historia pega otro per saltum, y hacia fines de la primera guerra mundial Herta estaba estudiando matemáticas en la Universidad de Ginebra, en Suiza. Mi abuela se llevó a la tumba las razones de tales cambios geográficos.

La abuela no me contaba mucho de su vida pasada. Tampoco le contó mucho a mi vieja, Evelyne, que en definitiva sabía bastante poco de su madre. Herta era una mujer inteligente, taciturna, muy culta. Leía constantemente, me explicaba los problemas de matemática que yo no entendía, y trató infructuosamente de enseñarme a jugar al ajedrez. Y yo, tan pelotudo fui, que nunca le presté atención.

La abuela egresó de la Universidad de Ginebra con honores: tuvo el honor de ser la primer mujer que se haya graduado en matemáticas en dicha institución. Su diploma, firmado por un tal William Rappard, estaba entre las cosas que descubrí en el departamento de mamá despues de su fallecimiento.

Después de graduarse, alrededor de 1921, Herta consiguió un trabajo en la recientemente creada Sociedad de las Naciones, en Ginebra; le tocó dar un examen de entrada, para el que aparentemente habia 5000 postulantes, y entraron mi abuela y dos personas más.

La abuela Herta era un cerebro.

Hablando de su estadía en Ginebra, una vez mi mamá encontró una de esas "agendas perpetuas" en las que mi abuela anotaba cumpleaños, direcciones y efemérides varias. En una de las páginas de aquella agenda, una inscripción misteriosa figuraba: "Place des Eaux-Vives, 19h, Tapio Voionmaa". Así nomás, sin el año ni nada. Mi vieja me contó que, al preguntarle sobre el tal Tapio, mi abuela estalló de ira (algo inusual en ella), le arrancó la agenda, y, le prohibió hablar de Tapio. La cosa quedó ahí; mi abuela se llevó el secreto al morir, pero mi mamá pensaba que habían sido novios, o quizás amantes. El tal Tapio fue, años más tarde, ministro y embajador de Finlandia.

Como verán, la historia de mi abuela está llena de hiatos y agujeros; sabrán disculpar la desprolijidad.

A fines de los años 20, antes del gran crash, mi abuela se casó (quizás á contre-coeur) con un tal Roland George, ingeniero mecánico de origen ginebrino.

La familia George había llegado a Ginebra desde Francia, durante las persecuciones contra los protestantes de las que se habla en los libros de historia. Ahí se instalaron, hicieron fortuna, y después la perdieron. Los descendientes de los George, entre los que se contaba a Roland, aparentemente lloraban día y noche por los tiempos pretéritos en que su familia poseía gran parte de lo que hoy se llama el Grand-Saconnex, barrio a 10 minutos del centro de Ginebra.

Hasta yo, de nene, escuché las historias de las riquezas que tenían los George. Cosas raras de la historia, mi primer departamento de soltero lo alquilé en el Grand-Saconnex, precisamente. Y mi vieja falleció en el barrio de al lado, el Petit-Saconnex, a 100 metros de las tumbas de su abuela y su tía, es decir, la mama y dos de las hermanas de la abuela Herta.

Los círculos se cierran de maneras misteriosas, a veces.

Escribo estas líneas mientras mi tren se detiene en la estación de Ginebra. Pareciese como si mi ADN tuviese un GPS integrado.

La abuela Herta se casó, entonces, con Roland hacia fines de la década del 20. Este Roland (que es el único de mis 4 abuelos al que nunca conocí personalmente) trabajaba para una empresa petrolera, llamada “Astra”. Esta empresa lo mandó a México, a principios de los años 30, para ocuparse de unos yacimientos. Fue allí que nació el primer hijo de Herta y Roland, mi tío Charles, allá por el año 32. 4 años después nació el segundo hijo, Henri, que se escribe con “i” y no con “y”, ya que es un nombre en francés. Ambos, pues, nacieron en México, creo yo en el DF, pero no estoy seguro.

Nunca conocí al tío Charles, ya que falleció en un accidente de auto en la ruta 2, en Argentina, en el año 62. Al tío Henri si lo conozco, pero no tengo mucho para decir.

A fines de los años 30, la empresa de mi abuelo lo mandó lo más lejos posible de Suiza (nunca supe si fue por causa de la guerra, o porque como todos me han contado, el tipo tenia un carácter insoportable). Fue así como la familia de mis abuelos terminó mudándose a Comodoro Rivadavia, en la provincia de Santa Cruz, en Argentina.

El culo del mundo, como decía mi abuela. No usaba seguido esa expresión; solamente al hablar de la Patagonia. Me contó que el viento la volvía loca. Al final, después de mucho insistir para irse de ahí, mis abuelos se radicaron en Buenos Aires. Corría el año 41 o 42.

Obviamente, para ese entonces, volver a Europa era impensable. Así que decidieron quedarse en Buenos Aires. Y en el 44 mi abuela se quedó embarazada por última vez, esta vez de mi madre, Evelyne. En aquella época la familia George-Schlerff vivía una vida anónima y apacible, en una casa de la calle Cuba en Belgrano, entre Quesada e Iberá, a pocas cuadras de la avenida Congreso.

De Filipópolis a Belgrano.

Mi abuelo Roland tenía una pequeña fábrica de repuestos para máquinas hidráulicas hacia fines de los 40, empresa que alimentaba la familia. Después, la empresa alimentaba también a los punteros peronistas del barrio, a los cuales se les pasaba un dinero para que los planes sociales de Eva Perón puedan prosperar, y para que no le cierren la fábrica o lo metan en cana.

Mi abuela nunca tuvo una inclinación política clara, salvo el hecho de ser una antiperonista acérrima. Me acuerdo lo alegre que estuvo cuando ganó Alfonsín en el 83, pero mi vieja me contó que también se alegró cuando fue el golpe del 76. La cosa era no estar a favor de Perón o de los peronistas.

La escena siguiente de la vida de mi abuela no es tan colorida o viajera. Por lo que pude entender y cotejar, mi abuela no había querido llevar a término el embarazo de mi madre; eso se tradujo en una cierta

apatía, que mi madre sufrió hasta el día de su muerte. Un abandono por parte de sus padres.

En realidad, no, peor que eso. Mi abuelo, en algún momento de la infancia de mi madre, abusó de su única hija. Mi madre descubrió ese recuerdo, totalmente hundido en su subconsciente, mediante terapia hipnótica.

Nunca supimos si mi abuela supo de ello o no, el tema es que en el año 62 se mata en la ruta 2 el hermano mayor de mamá, Charles. Este acontecimiento es una bisagra, un punto de inflexión que modificó para siempre la dinámica de la familia. Charles era el hermano amado de mi madre, el que la protegía, el que, quizás, supo la verdad del abuso.

Mi madre, sin embargo, supo de la muerte de su hermano a través de un periódico. Leyendo, por casualidad, la sección de avisos fúnebres. Porque, por alguna razón ignota, mi abuela no le quiso avisar la noticia a mamá, que se enteró de todas maneras, de esta manera, horrible si las hay.

La locura humana no tiene raíces inexplicables. A veces son incomprensibles, pero no son nunca inexplicables. Mi vieja vaciló en el precipicio de la locura más absoluta en ese preciso instante. Toda la vida de mi madre, toda su relación con la abuela Herta, y también su relación conmigo, fue definida en ese preciso instante.

Mi abuela se separa de Roland a principios de los 60, y con mamá se van a vivir a una casa sobre la calle Haedo, en el barrio de Vicente López, a unas cuadras del cruce con la avenida Maipú, del lado de Florida. Mi abuela laburaba de traductora para una empresa mecánica de origen alemán, que tenía sus oficinas en Munro. Mi vieja laburaba de vendedora, y trataban de llegar a fin de mes, muchas veces comiendo la misma polenta que le preparaban a la perra.

Hablando de idiomas, para esa época mi abuela Herta le enseñaba francés a una amiga entrañable de mi mamá, Eleonora, que tenía que preparar los exámenes de la Alianza Francesa. Años más tarde, antes de venir a vivir a Suiza, allá por el 89, Eleonora me enseñó el francés a mí. Y tal vez, en estos momentos, una de las hijas de Eleonora esté leyendo este texto, y así va la vida.

Finalmente, hacia fines de los 60, mi abuela Herta y mi mamá lograron ahorrar algo y se compraron un departamento con vista al río, en la zona del bajo Vicente López; sobre Avenida Libertador, entre San Martín y Arenales. El arquitecto que estaba a cargo de las obras era un pibe pintón de ojos verdes de unos 27 años llamado Alberto Kosmaczewski. Al poco tiempo, mi mamá y él empezaron a salir juntos, y se casaron en el 71.

Exactamente para esa época, mi abuela tuvo su primer infarto. Fue un domingo a la tarde, después del almuerzo; mi mamá y ella estaban viendo “Los Campanelli” en la televisión, y mi abuela le dijo a mi vieja “me siento mal”, y entró en coma.

Tuvo tres paros cardio-respiratorios en pocas horas, y estuvo en coma 21 días. Milagrosamente se salvó, y tan buena fue su recuperación que se fue a Europa a visitar a sus hermanas y familia.

Cabe indicar que para esa época la situación financiera de la familia se había mejorado mucho. Mi abuela empezó a cobrar la jubilación suiza, que dado el tipo de cambio de aquella época, representaba un ingreso comparable al del un senador, un narcotraficante y un gerente de multinacional, todos juntos y multiplicado por 2.

En setiembre del 73 nací yo. La abuela me adoraba; me traía regalos de Europa y así crecí, con ella y mi mamá. Tal vez mi llegada haya servido para que la relación entre ambas se endulce; en todo caso, no tengo recuerdos de peleas jodidas entre ellas. Si, una cierta tensión, palpable, pero ni gritos ni platos rotos.

Pensándolo bien, tal vez les hubiese venido bien romper algunos platos. No sé.

En 1981 le diagnosticaron la enfermedad de Parkinson a mi abuela Herta. A partir de ahí, su estado de salud empezó a desmoronarse inexorablemente. Temblaba mucho, babeaba continuamente. En el 83 la operaron del nervio trigémino, y eso le curó el Parkinson, y fuimos a festejar sus 80 años a la costa atlántica. Pero tanto tratamiento la dejó muy débil.

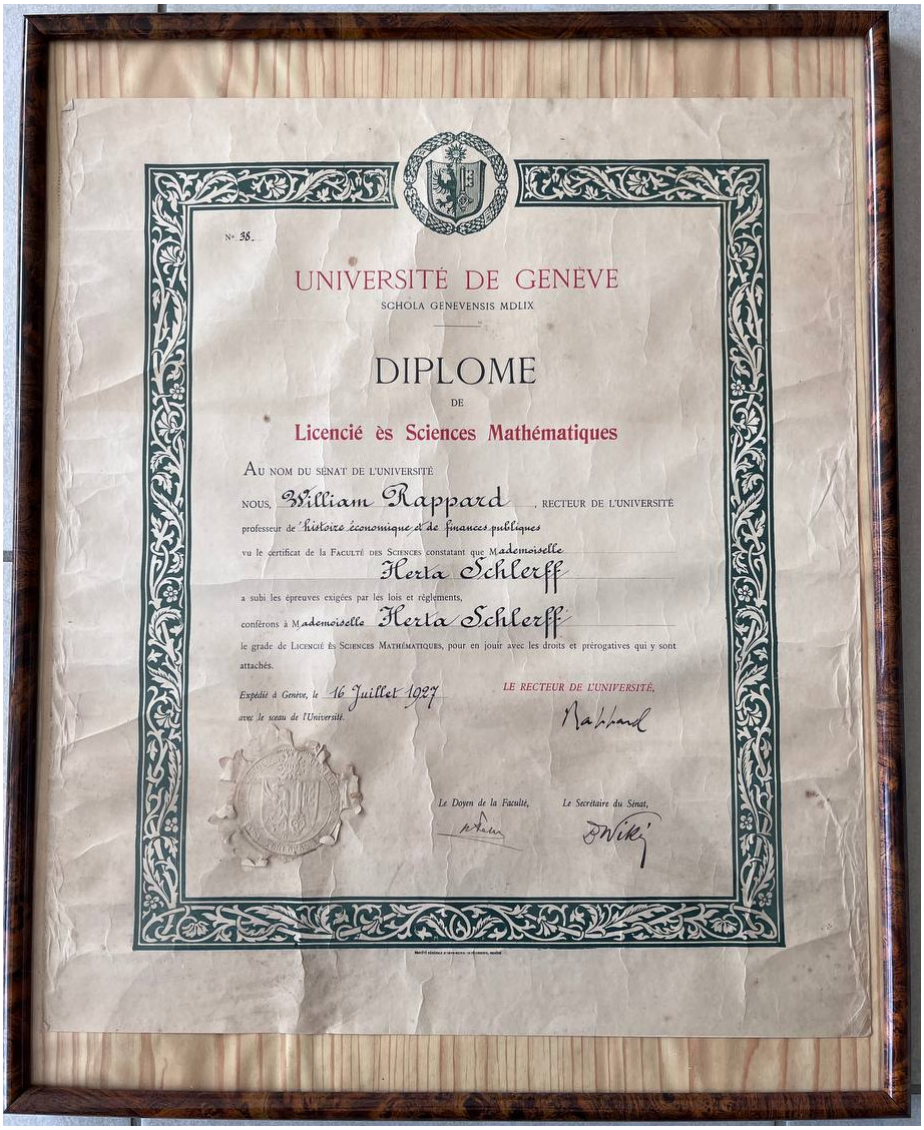
Festejamos su último cumpleaños una noche de diciembre del 84. Tenía 81 años. Hacía un calor de locos, me acuerdo, y ella sufría mucho el calor.

A principios de abril del 85 se le fracturó la cadera y se cayó; lo que no supieron los médicos fue si se le fracturó la cadera por caer, o si se cayó por que se le fracturó la cadera; ellos nos decían que ambas situaciones eran posibles, aunque en realidad, el dato es de una inutilidad espantosa. La operaron y le pusieron una prótesis, y 10 días después falleció.

La última vez que la vi fue un viernes, después de la escuela. Yo iba a la numero ocho, que está enfrente de la clínica. Me acuerdo que cuando nos íbamos de su habitación justo llegaban unos médicos para revisarla, y ella me miraba fijo mientras se cerraba la puerta, y yo caminaba, de la mano de mi vieja, mirando para atrás y siguiendo su mirada.

Esa mirada, su mirada se quedó para siempre en mi memoria.

Update, 2023-05-12: Una foto de su diploma original:



N° 58.

UNIVERSITÉ DE GENÈVE

SCHOLA GENEVENSIS MDLIX

DIPLOME

DE

Licencié ès Sciences Mathématiques

Au nom du Sénat de l'Université

nous, William Rappard, recteur de l'Université

professeur de histoire économique et de finances publiques

vu le certificat de la Faculté des Sciences constatant que Mademoiselle

Herta Schlerff

a subi les épreuves exigées par les lois et réglements,

conférons à Mademoiselle Herta Schlerff

le grade de Licencié ès Sciences Mathématiques, pour en jouir avec les droits et prérogatives qui y sont attachés.

Expédié à Genève, le 16 Juillet 1927

avec le sceau de l'Université.

LE RECTEUR DE L'UNIVERSITÉ.

Rappard

Le Doyen de la Faculté,

Le Secrétaire du Sénat,

This week: Mobilism 2012 Amsterdam

Adrian Kosmaczewski

2012-05-07

This week we are very happy to announce that will be attending the 2012 edition of Mobilism Amsterdam[1] organized by Peter-Paul Koch (of QuirksMode[2] fame), Krijn Hoetmer and Stephen Hay!

[3]

From the Mobilism site:

> Mobile is becoming increasingly important to web designers and developers because users expect a site to work on their phones. Simultaneously, the web is becoming increasingly important to the mobile world because it is the only way to deploy an application to any phone.
>
> Nowadays most web conferences feature a mobile session, and most mobile conferences a web session. The obvious next step is Mobilism: a conference wholly dedicated to mobile web design and development.
>
> Just like in 2011 we'll invite some of the best speakers[4] from the web development and the mobile world to guide you through the confusing jumble of platforms, screen sizes, and browsers that is the mobile ecosystem.

Check out the list of speakers[5]: Horace Dediu, James Pearce, Remy Sharp, Jeremy Keith, Brian LeRoux and more!

If you are around[6] let's meet and talk about the mobile web.

[1] http://mobilism.nl/2012
[2] http://quirksmode.org/
[3] http://mobilism.nl/2012
[4] http://mobilism.nl/2012/programme
[5] http://mobilism.nl/2012/programme
[6] http://mobilism.nl/2012/attendees

Introducing the Henri Dès iPhone App: La Radio HD!

Adrian Kosmaczewski

2012-05-14

We are extremely happy to announce the immediate availability of the Henri Dès iPhone application[1] on the App Store. This is the result of a collaboration with our dear friends Fabien and Sena from We Studio[2] Lausanne, and is a mobile frontend for the Radio Henri Dès website[3], also created by We Studio[4].

[5]

This application streams more than 250 songs in high definition, without ads or interruptions, non-stop 24/7! It also provides quick access to the songs by Henri Dès in your iOS device, and shortcuts to buy more from the iTunes store.

Henri Dès[6] is a Swiss singer and songwriter immensely popular with children in francophone countries. He represented Switzerland in the 1970 Eurovision Song Contest with the song “Retour”, earning the fourth position. Publishing since 1977 under his own label, Marie-Josée Productions, he has written, recorded and released 25

[1] http://itunes.apple.com/ch/app/la-radio/id524936258?l=en&mt=8
[2] http://www.we-studio.ch/
[3] http://radio-henrides.net/
[4] http://www.we-studio.ch/
[5] http://itunes.apple.com/ch/app/la-radio/id524936258?l=en&mt=8
[6] http://itunes.apple.com/ch/artist/henri-des/id39893482?l=en

records, and has toured extensively in Switzerland, France, Belgium and Canada.

We are very happy, please go and check it out on the App Store[7]!

[8]

[7] http://itunes.apple.com/ch/app/la-radio/id524936258?l=en&mt=8
[8] http://itunes.apple.com/ch/app/la-radio/id524936258?l=en&mt=8

A Mobile Developer Lab the Size of a Country!

Adrian Kosmaczewski

2012-05-17

Last week we attended with Bertrand Dufresne[1] the 2012 edition of Mobilism[2] in Amsterdam, an awesome event set up by many great people, among them Peter-Paul Koch[3], Stephen Hay[4] and Krijn Hoetmer[5].

One of the most inspiring and useful ideas we brought from Mobilism was the creation of a mobile testing lab, covering a wide array of devices, allowing companies of all sizes to offer access to devices from different brands and models. This idea was presented by both Remy Sharp[6] and Jeremy Keith[7], and it has taken the form of a physical testing lab set up in Brighton, UK, as described by Jeremy himself[8].

[1] https://twitter.com/bdufresne
[2] http://mobilism.nl/2012
[3] http://quirksmode.org/
[4] http://www.the-haystack.com/
[5] http://krijnhoetmer.nl/
[6] http://remysharp.com/
[7] http://adactio.com/
[8] http://adactio.com/journal/5446/

Of course, Switzerland is a country, and slightly bigger than Brighton, but as a country, it is not that big after all. Imagine an online database of devices, including their geographical location, helping developers to share devices with each other, to reserve time slots, and this not only in Geneva or Zürich, but all over the country.

Very often we developers would love to be able to test our latest app in some weird combination of mobile OS or browser or technology, and we wonder whether someone has such a device to lend for a little while, the time of debugging our code or testing our site. That's the whole spirit of this project.

So far the idea, thus, is to start the discussion and start collecting information. Towards this goal, I've opened a project site[9] to share conversations, files and other items of interest around this idea.

If you want to join this effort, please contact me[10] and I'll open a account for you to the project. We would love to have a large array of devices for testing, and if we all collaborate, we could have a country-wide database of devices! I think this is a goal that will provide lots of utility for all of us.

[9] http://projects.akosma.com/projects/open-mobile-device-lab
[10] /about/

Astucia Ferroviaria

Adrian Kosmaczewski

2012-05-31

Conozco los trenes suizos bastante bien como para tener toda una serie de trucos que me permiten viajar mas cómodo, mas rápido, mas tranquilo.

Algunos son simples y pelotudos, como por ejemplo evitar los trenes de hora pico. Bueno, hasta ahí, nada nuevo.

Pero acá va algo que seguramente no saben; todos los trenes de Suiza (y cuando digo todos son todos) tienen los vagones de primera clase mirando hacia Zurich. Todos.

Me explico; los trenes suizos son unos convoys de la hostia. Imaginense bestias de 14 vagones de 30 metros de largo cada uno, si, 400 metros de trenes con dos pisos que van a 160 km/h entre Ginebra, Berna y Zurich. Las estaciones en las cuales se detienen estas moles son consecuentes. Particularmente las de Berna y Zurich, que son las mas importantes del trayecto.

Estos trenes tienen vagones de primera y segunda clase, que solo difieren por la cantidad de espacio para las piernas. Pero los vagones de primera clase, invariablemente, están del lado del tren que mira para Zurich.

Por que es esto así? Bueno, resulta que la estación de Zurich es la única en cul de sac, es decir, es la única en la cual los trenes no siguen de largo, sino que tienen que salir marcha atrás, por el mismo camino que llegaron. De esa manera, los pasajeros que estan en primera siempre caminan lo menos posible para llegar o salir del tren.

Ventajas de la clase.

Todas las estaciones de suiza estan preparadas para este hecho, y por ejemplo, las paradas de taxis, los mejores restaurantes y los pasillos mas cómodos, están colocados justamente del lado donde frenan los vagones de primera clase.

Talking at SwissJeese 2012

Adrian Kosmaczewski

2012-06-01

I'm very proud and happy to announce that tomorrow I'll be talking in the first edition of SwissJeese[1]!

This is the national Swiss JavaScript conference, a free event organized by Nelmio[2] and Bertrand Dufresne[3], with the sponsorship of webdoc[4] and the help of Relax in the Air[5], who have contributed the website.

I'm going to be talking about Sencha Touch[6], of course! I hope to see you all there tomorrow. The event is free but requires registration[7], so don't hesitate, sign up and let's meet! SwissJeese 2012 will happen tomorrow in PROGR Bern[8] from 10 AM to 8 PM.

[1] http://2012.swissjeese.com/
[2] http://nelm.io/
[3] https://twitter.com/bdufresne
[4] http://www.webdoc.com/
[5] http://www.relaxintheair.com/
[6] http://www.sencha.com/products/touch/
[7] https://swissjeese.eventbrite.com/
[8] http://www.progr.ch/

SwissJeese

SwissJeese 2012 Slides, Code and Pics

Adrian Kosmaczewski

2012-06-04

Last Saturday took place the first edition of SwissJeese[1] in Bern! The event was an outstanding success and kudos to the organizers for an incredibly cool gathering.

For those interested, I've published in Github the source code of the application[2], as well as the slides of the presentation[3] on Speaker Deck (these were created using Paper for iPad[4] with a Cosmonaut Stylus[5], if you were wondering.) Finally, you can check out some photos of the event[6] in our Facebook page.

Enjoy!

[1] http://2012.swissjeese.com/

[2] https://github.com/akosma/Sencha-Touch-2-Minimal-MVC/

[3] https://speakerdeck.com/u/akosma/p/intro-to-sencha-touch-2

[4] http://www.fiftythree.com/paper

[5] http://www.studioneat.com/products/cosmonaut

[6] https://www.facebook.com/media/set/?set=a.10150945332221100.442216.184046196099&type=3

- - -

title: ‘Attending WWDC 2012’ date: 2012-06-08 author: ‘Adrian Kosmaczewski’ draft: false tags: [‘akosma software’, ‘ios’, ‘conferences’]
—

Next week I’ll be attending the 2012 edition of WWDC in San Francisco[1]! A whole week full of training sessions, labs, and meetings with cool friends from all over the world. And apparently with many surprises, as several sessions in the schedule[2] (requires login) are marked as “to be announced”.

Just like in 2008[3], 2009[4] and 2010[5], I will be covering this event on this blog, providing details... of the keynote, which is the only part that I can safely talk about :)

For those of you going there for the first time, check out the always excellent WWDC guide by Jeff LaMarche[6], updated for the 2012 edition. If we meet in the hallways, don’t hesitate to stop me to say hi! I’d love to meet you in person. WWDC is much more than a conference, is a gathering of very cool people, real artists who ship amazing applications. Don’t forget to get your copy of the official WWDC app[7] in your iPhone or iPad! (oh, and check the latest “WWDC tips” by Jeff Johnson[8] in his Twitter timeline, they are hilarious... and so true :)

Personally I am very happy to meet many friends like Jørn Larsen[9], Joe

[1] https://developer.apple.com/wwdc/
[2] https://developer.apple.com/wwdc/schedule/
[3] /blog/i-was-there/
[4] /blog/wwdc-2009/
[5] /blog/wwdc-2010/
[6] http://iphonedevelopment.blogspot.ch/2012/05/wwdc-first-timers-guide-2012-edition.html
[7] http://itunes.com/wwdc12
[8] https://twitter.com/lapcat
[9] http://www.trifork.ch/

d'Andrea[10], Daniel Steinberg[11], Daniel Pasco[12], Danilo Campos[13], and of course the amazing group of 10 members of the immedia team[14], led by Anice Hassim and Nazeem Ebrahim who arrived yesterday to the US from Durban, South Africa!

Finally, check out the official list of WWDC parties![15] I will be attending (at least) the iOS Intelligence Party[16] by Raven Zachary on Monday and the WWDC Bash on Thursday, so if you are around let's meet and share a beer.

This WWDC will be epic!

[10] http://www.linkedin.com/in/jdandrea
[11] http://dimsumthinking.com/
[12] http://blackpixel.com/
[13] http://blog.danilocampos.com/
[14] http://instagr.am/p/LlaFWgSptt/
[15] http://wwdcparties.com/
[16] http://raven.me/wwdc12party/

Anticuario De La Web

Adrian Kosmaczewski

2012-06-08

Ya soy un programador viejo. Muchas veces me encuentro delante de otros programadores y les cuento, que cuando empecé creando páginas web en el ’96…

… en aquellos tiempos usábamos HoTMetaL Pro o el editor de Netscape 3…

… que teníamos que hacer las páginas compatibles con el `<LAYER>` de Netscape y el `<DIV>` del Internet Explorer…

… que el CSS era una curiosidad del Internet Explorer, y que la norma era usar el atributo `FONT` y otras atrocidades similares…

… que la mayor parte del código HTML que escribíamos era en mayúscula, y que los parámetros de los atributos muchas veces venían sin comillas…

… que JavaScript era totalmente incomprendido en aquellos tiempos, lejos estábamos de tener algo como jQuery, y que lo mas parecido a un JavaScript cross-browser era lo que salía de las entrañas de Dreamweaver…

… que escribir componentes ActiveX o applets en Java era más común (y hasta fácil) que crear movies en Flash…

… que Opera era un pedo en el océano…

… que Apple estaba a punto de quebrar…

… que era común que un formulario utilice el atributo `METHOD=GET`…

… que el UTF-8 era un mito…

… y me miran con ojos de huevo frito, atónitos, incrédulos y mirando para otro lado.

Announcing my first book: “Mobile JavaScript Application Development”

Adrian Kosmaczewski

2012-06-14

Today I’m announcing my first book, “Mobile JavaScript Application Development”[1], available for sale in print and in DRM-free electronic formats (PDF, ePub, Kindle, etc.). The book is adapted from the mobile web app trainings I’ve been giving in Zurich[2] and South Africa[3], about JavaScript, HTML5, jQuery Mobile, Sencha Touch and PhoneGap.

Buy your copy now at any of these shops!

- O’Reilly[4]
- iBooks Store (US only)[5]

[1] http://shop.oreilly.com/product/0636920025252.do
[2] http://mobile-training.ch/
[3] http://www.appdev.co.za/
[4] http://shop.oreilly.com/product/0636920025252.do
[5] http://itunes.apple.com/us/book/mobile-javascript-application/id537701148?mt=11

- Amazon.com[6]
- Amazon.co.uk[7]
- Amazon.ca[8]
- Amazon.de[9]
- Amazon.fr[10]
- Amazon.co.jp[11]
- Amazon.es[12]
- Amazon.it[13]

Or read it through O'Reilly Safari Books Online[14]!

It is a compendium, an introduction, and a "state of the union" kind of book, that will help (I hope) desktop and web developers wanting to get into the mobile web train as fast as possible. It is not a "bible" book, but rather a series of introductions, with sufficient detail and explanations to be able to get up and running, fast, on these new exciting technologies.

But there is another detail about this book; I've written it in English, which, well, as you know is not my mother tongue. This makes it all the more special to me.

This book would not have been possible without the help and comments from Simon St. Laurent, my editor at O'Reilly; Maximiliano Firtman, Jens-Christian Fischer, Gabriel García Marengo, Bertrand Dufresne, Anice Hassim, Kishyr Ramdial, Mats Bryntse, and of course, the continuous support of my dear Claudia, without whom all life would be sheer unhappiness and utter impossibility.

The code samples of the book can be found, as usual, in Github[15]. Feel free to fork, share and enjoy!

Some dreams do come true. I am very happy today, and I sincerely hope that this book will bring some "aha!" moments around the world. Enjoy, and I'll be glad to get any feedback for, who knows, a second edition some time in the future.

[6] http://www.amazon.com/Mobile-JavaScript-Application-Development-Kosmaczewski/dp/1449327850

[7] http://www.amazon.co.uk/Mobile-JavaScript-Application-Development-Kosmaczewski/dp/1449327850

[8] http://www.amazon.ca/Mobile-JavaScript-Application-Development-Kosmaczewski/dp/1449327850

[9] http://www.amazon.de/Mobile-JavaScript-Application-Development-Kosmaczewski/dp/1449327850

[10] http://www.amazon.fr/Mobile-JavaScript-Application-Development-Kosmaczewski/dp/1449327850

[11] http://www.amazon.co.jp/Mobile-JavaScript-Application-Development-Kosmaczewski/dp/1449327850

[12] http://www.amazon.es/Mobile-JavaScript-Application-Development-Kosmaczewski/dp/1449327850

[13] http://www.amazon.it/Mobile-JavaScript-Application-Development-Kosmaczewski/dp/1449327850

[14] http://my.safaribooksonline.com/9781449327842?portal=oreilly&cid=orm-cat-readnow-9781449327842

[15] https://github.com/akosma/Mobile-JavaScript-Application-Development

Update, 2012-07-17: Here's the official press release by O'Reilly[16] announcing the release of the book.

Update, 2012-08-15: Check out the overwhelmingly positive reception[17] to the book so far!

[16] http://oreilly.com/pub/pr/3070

[17] /blog/praise-for-mobile-javascript-application-development/

Introducing the Lausanne Cathedral iPhone Applications!

Adrian Kosmaczewski

2012-07-03

We are thrilled to announce the immediate availability of the Lausanne Cathedral iPhone application on the App Store, in French[1], English[2] and German[3]!

[4]

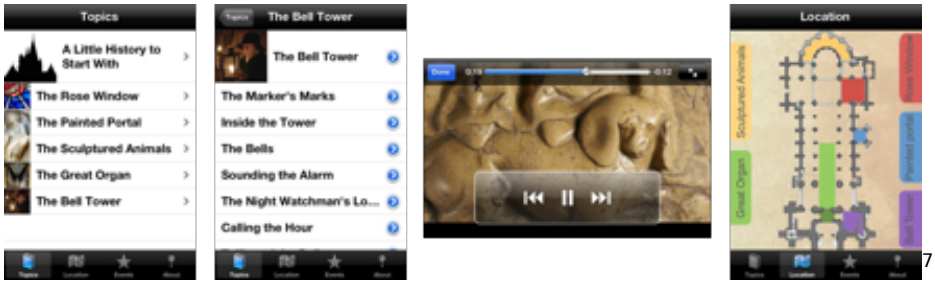
These applications are the result of a long collaboration with historians, philosophers, video producers and media experts under the direction of the EERV[5], who have worked together to create three stunning applications, providing insight and useful information about the Cathedral[6].

It will be useful to tourists, visitors, academia and the general public to know more about this beautiful building, its story, architecture and other details. As explained on the App Store:

> The Eglise Réformée Vaudoise, the EERV, that is the Protestant Church of the Canton of Vaud has created an application to help visitors find their way through the cathedral. Its

[1] http://itunes.apple.com/au/app/cathedrale-de-lausanne/id533055625?mt=8
[2] http://itunes.apple.com/au/app/lausanne-cathedral/id533081182?mt=8
[3] http://itunes.apple.com/au/app/kathedrale-von-lausanne/id533076641?mt=8
[4] http://itunes.apple.com/au/app/lausanne-cathedral/id533081182?mt=8
[5] http://eerv.ch/
[6] http://lacathedrale.eerv.ch/

purpose is to allow them maximum autonomy, and to render their experience meaningful, as this historical edifice is also a place of high level of spirituality.



[7]

Check them out on the App Store[8]!

[7] http://itunes.apple.com/au/app/lausanne-cathedral/id533081182?mt=8
[8] http://itunes.apple.com/au/app/lausanne-cathedral/id533081182?mt=8

Mobile App Dev Training in South Africa

Adrian Kosmaczewski

2012-07-05

Just like in October 2011[1] and February 2012[2], we'll be back in South Africa the next 3 weeks, for a whole series of mobile application trainings[3] organized jointly by immedia[4] and akosma software!

[5]

Have you ever wanted to learn how to create a mobile app? Looked at Evernote and thought, "Hey, I could do that!" We have embarked on an exciting training initiative aimed at providing South African businesses and individuals with the skills they need to become global app creators.

Courses[6] will be held in Cape Town, Johannesburg and Durban, and will be provided[7] by Kishyr Ramdial, specialist cloud and mobile app developer, and myself:

1. iOS Beginners[8]: 2-day course perfect for: Beginner developers, who have working knowledge of Visual Basic, C, C++, C#, Java, and students with development experience wanting to specialise in iOS Development.
2. iOS Advanced[9]: 3-day course perfect for: Senior Developers looking for the next big challenge and iOS Developers wanting to enhance their skills.
3. Mobile Web App Development[10]: 3-day course perfect for developers who need to publish an app quickly to all platforms.

[1] /blog/mobile-application-training-in-south-africa/
[2] /blog/more-mobile-application-training-in-south-africa/
[3] http://www.appdev.co.za/
[4] http://www.immedia.co.za/
[5] http://www.appdev.co.za/
[6] http://www.appdev.co.za/courses/
[7] http://www.appdev.co.za/about/trainers/
[8] http://www.appdev.co.za/courses/iosbeginners/
[9] http://www.appdev.co.za/courses/iosadvanced/
[10] http://www.appdev.co.za/courses/crossplatformappdev/

 [11]

There are courses targeted at app developers, dev beginners, and executives – to ensure that you learn exactly what you need to join this app revolution!

Check out the complete program here: <http:></http:> and sign up contacting us at <http:></http:>, but hurry up! There are very few spots left.

[11] http://www.appdev.co.za/

Mobile JavaScript Application Development Book Launch Event in South Africa

Adrian Kosmaczewski

2012-07-17

I'm very glad to announce that our friends of [immedia] in Durban, South Africa, will be hosting a launch event[1] for our latest book[2] "Mobile JavaScript Application Development"! It will take place next Thursday at 6 PM in the immedia headquarters[3] (Level 3 @ The Quarterdeck, 69 Richefond Circle, Ridgeside Office Park, Umhlanga 4320, Durban).

Read more at their blog[4] and we hope to see you there!

[1] http://www.immedia.co.za/featured-content/book-launch-mobile-javascript-application-development/

[2] /blog/announcing-my-first-book-mobile-javascript-application-development/

[3] http://www.immedia.co.za/contact-us/

[4] http://www.immedia.co.za/featured-content/book-launch-mobile-javascript-application-development/

Praise for “Mobile JavaScript Application Development”

Adrian Kosmaczewski

2012-08-15

Si Dunn, at Books, Books & More (New) Books[1]:

> His book is divided into seven well-written chapters. And six of them offer numerous screenshots and short code examples. (…) Mobile JavaScript Application Development takes this straightforward approach: (1) “leave the theory to others” and (2) focus on “understand by doing.” And, mercifully, the author does not try to tackle too many technologies at once. Instead, he concentrates – in “an opinionated, hands-on” way on three technologies that he says “are currently the most promising and…show the most interesting roadmap.”

S. Shanbhag, at Amazon.com[2]:

[1] http://sagecreek.wordpress.com/2012/07/11/mobile-javascript-application-development-bringing-the-web-to-mobile-devices-programming-bookreview/

[2] http://www.amazon.com/Mobile-JavaScript-Application-Development-Programmin

> Overall, this book is very pleasant to read and is really geared towards folks evaluating different JavaScript mobile frameworks and didn't know where to start. This book will make it easy for you to decide what's suitable for your application.

Juri Strumpflohner, at Juri's TechBlog[3]:

> If you are a JavaScript newbie eager to dive into creating rich mobile web applications, this book is for you. It is structured in a very clear and intuitive way, introducing the most useful HTML5 features, some JavaScript performance tips, jQuery mobile and Sencha Touch examples and illustrates even how PhoneGap works.

Elissa Shevinsky, at the O'Reilly book page[4]:

> This book is well organized and easy to follow with clear instructions for using these technologies to build basic applications. While I can enthusiastically recommend this book as an excellent starting point, further reading is necessary to build an app with high performance and seamless UI.

Piers Hollott, at the O'Reilly book page[5]:

> I have just finished reading a review copy of Adrian Kosmaczewski's book on Mobile Development using JavaScript, and I highly recommend it, particularly if you are faced with a decision about mobile development frameworks and you have a team which is already familiar with hybridized JavaScript approaches like JQuery or GWT.

Buy your copy and leave us your review of "Mobile JavaScript Application Development" now at any of these fine locations:

- O'Reilly[6]
- iBooks Store (US only)[7]
- Amazon.com[8]
- Amazon.co.uk[9]

---

g/product-reviews/1449327850/ref=cm_cr_dp_see_all_btm?ie=UTF8&showViewpoints=1&sortBy=bySubmissionDateDescending

[3] http://blog.js-development.com/2012/08/mobile-javascript-application.html?utm_source=twitterfeed&utm_medium=twitter&utm_campaign=Feed%3A+juristrumpflohner+%28Juri+Strumpflohner%27s+Blog%29

[4] http://shop.oreilly.com/product/0636920025252.do#PowerReview

[5] http://shop.oreilly.com/product/0636920025252.do#PowerReview

[6] http://shop.oreilly.com/product/0636920025252.do

[7] http://itunes.apple.com/us/book/mobile-javascript-application/id537701148?mt=11

[8] http://www.amazon.com/Mobile-JavaScript-Application-Development-Kosmaczewski/dp/1449327850

[9] http://www.amazon.co.uk/Mobile-JavaScript-Application-Development-Kosmaczewski/dp/1449327850

- Amazon.ca[10]
- Amazon.de[11]
- Amazon.fr[12]
- Amazon.co.jp[13]
- Amazon.es[14]
- Amazon.it[15]
- O'Reilly Safari Books Online[16]
- Source code of the book in Github[17]
- Goodreads page of the book[18]

[10] http://www.amazon.ca/Mobile-JavaScript-Application-Development-Kosmaczewski/dp/1449327850

[11] http://www.amazon.de/Mobile-JavaScript-Application-Development-Kosmaczewski/dp/1449327850

[12] http://www.amazon.fr/Mobile-JavaScript-Application-Development-Kosmaczewski/dp/1449327850

[13] http://www.amazon.co.jp/Mobile-JavaScript-Application-Development-Kosmaczewski/dp/1449327850

[14] http://www.amazon.es/Mobile-JavaScript-Application-Development-Kosmaczewski/dp/1449327850

[15] http://www.amazon.it/Mobile-JavaScript-Application-Development-Kosmaczewski/dp/1449327850

[16] http://my.safaribooksonline.com/9781449327842?portal=oreilly&cid=orm-cat-readnow-9781449327842

[17] https://github.com/akosma/Mobile-JavaScript-Application-Development

[18] http://www.goodreads.com/book/show/13595163-mobile-javascript-application-development

Cross-Platform Mobile Web Application Development Training in London

Adrian Kosmaczewski

2012-08-23

We are very happy to announce the first edition of the Cross-Platform Mobile Web Application Development Training in London[1]!

Today, having a mobile application online is a "must." But there are multiple platforms to write for, each with their own language, idioms and pitfalls. Luckily there is a simple solution that allows to write once and deploy on all modern mobile devices: HTML5 and JavaScript.

What

This three day intensive course, brought to you jointly by Zerofee[2] and akosma software[3] takes you from being a web developer to being a mobile developer. We take you through the basics of writing HTML5 applications for mobile devices, cover the additional APIs that allow you to access the functions of the devices (like storage, geo-location, accelerometers) and put you in control of deploying an application to either iOS or Android devices.

The training will cover the following:

- Overview of options for bringing web apps to a mobile device
- Overview of HTML5 technologies for mobile development
- JavaScript best practices
- In depth review of jQuery Mobile and Sencha Touch
- Bundling your app for the device with PhoneGap
- Accessing your devices sensors and special features from JavaScript

What you will walk away with

There will be lots of hands-on working building an application from scratch and bringing it to life on your mobile device. We will build two applications – one with jQuery Mobile, one with Sencha Touch and bring them to "life" as native applications on a mobile device.

[1] http://mobilewebapplondon.eventbrite.com
[2] http://zerofee.org
[3] http://akosma.com

- 3 days of intensive, hands-on training in a small group (max 12 people), with plenty of time to talk to the instructor.
- A complimentary copy of the Mobile JavaScript Application Development[4] book.
- Full lunch meal.
- Plenty of snacks and drinks during the day.

How

To attend this training, you must have prior experience in the following technologies:

- HTML, CSS
- Basic JavaScript skills
- Previous programming experience

These are the hardware and software requirements:

- Laptop (Mac or PC) with either iOS SDK (OS X with XCode) or Android SDK (OS X, Windows, Linux) installed and running
- Any text editor
- Web browser: Safari or Chrome

Optionally, you can also bring your mobile device (iOS, Android, Windows Mobile) to install your mobile web apps into it.

Who

Adrian Kosmaczewski is the author of the book Mobile JavaScript Application Development[5] recently published by O'Reilly. He has been writing software for the past 20 years, riding the web since the time Netscape 2 was the next big thing. He started writing Cocoa applications for the Mac in 2002, and has been writing iOS apps since he returned from WWDC 2008. Adrian is the founder of akosma software[6], with a strong focus in all things web and iOS.

Adrian is an experienced trainer in IT and programming, and he has already offered this training in Switzerland[7] and South Africa[8] with great success.

When and Where

The 3 day course will be held from October 1st to October 3rd in London, in the Hoxton Hotel, 81 Great Eastern Street, EC2A 3HU London[9].

[4] http://shop.oreilly.com/product/0636920025252.do

[5] http://shop.oreilly.com/product/0636920025252.do

[6] http://akosma.com/

[7] http://mobile-training.ch

[8] http://www.appdev.co.za

[9] https://maps.google.com/maps?q=The+Hoxton+Hotel+81+Great+Eastern+Street+EC2A+3HU+London+United+Kingdom&hl=en&client=safari&hq=The+Hoxton+H

Questions?

Just send us an email to ela@zerofee.org[10] or adrian@akosma.com[11] for any question you may have.

otel+81+Great+Eastern+Street+EC2A+3HU+London+United+Kingdom&radius=15000&t=m&z=16

[10] mailto:ela@zerofee.org

[11] mailto:adrian@akosma.com

Plan for a Brighter Smile

Adrian Kosmaczewski

2012-08-28

Every so often I decide to make what could visually be described as "cutting the fat" in my life.

Taking out elements that make me heavy, that do not provide any enjoyment, that drag me down, that I probably used to enjoy in a past life, but that do not bring any pleasure anymore.

Developing apps for third parties was one of them, definitely, and this is what motivated my decision of stopping that part of my business[1], giving me the energy and CPU time to concentrate in the parts that I'm enjoying the most: consulting and training.

I'm not going to jump into a rant like those I've wrote in the past[2], because I think I've been fortunate enough to participate, albeit in a small, probably inconsequential way, to the 3rd or 4th biggest industrial revolutions known to mankind. First the web, now the mobile; I'm happy to have been at the right time, at the right place. I met incredible people, many of which have changed my life forever in subtle, uncanny, and sometimes even earth-shattering ways.

However, there are factors in the services business that are, simply put, unbearable. Getting to sign NDAs to discover later how bad some ideas are; having to explain once and again that the sale price of an app has absolutely no connection whatsoever with the development cost; dealing with non-technical middlemen who will weigh their political influence to get their mindless input into the final product; und so weiter.

I am tired of all that.

What now? Well, the future. The bright and beautiful future. I am going to expand my teaching operations; simply because that's the thing I enjoy the most. I enjoy being able to transmit to others what I've learnt. And, of course, writing more books is part of the deal. I simply need to write, I need to feel my fingers on the keyboard, sewing stitches of knowledge, all while sipping a maté[3] or listening to some good old progressive rock. I'm also going to work on my own apps, of course.

[1] http://akosma.com/

[2] /blog/i-hate-you-airline-industry/

[3] /blog/on-the-importance-of-yerba-mate-in-the-software-development-process/

Maybe at some time I'll dive into other kinds of writing, like fiction or comedy (I've been wanting to write a one-man show for a while), write books in other languages too, start a podcast, star in a movie, play the piano again. Who knows. I've got a right side of my brain that's mostly underexplored, and it's screaming to get out.

I'll be around, of course; it's just that, well, from now on I'll have a brighter smile on my face.

Les Gens Pensent Que Je Suis Tessinois - Part I

Adrian Kosmaczewski

2012-09-04

C’est sérieux, c’est toujours pareil. Apparemment j’ai un accent étrange, relativement difficile à placer, paraît-il. Alors les gens me demandent, “alors, vous venez du Tessin?”

Ben oui, vous voyez mon nom de famille? Je fais partie des fameux tessinois polonais, les “Luganiski”. On mange des pierogis farcis au risotto, c’est mortel.

Evidemment la question qui suit c’est “et vous venez d’où déjà?”, parce qu’il est vachement important de savoir d’où on vient. On parle différemment aux gens suivant leur origine. Non, mais c’est vrai. Moi par exemple lorsque je me retrouve devant des argentins je leur parle en espagnol, voyez vous. Si je leur parlait en français ils me regarderaient avec un oeil éberlué qui varierait de l’émerveillement a l’incompréhension la plus absolue.

A moins qu’ils ne parlent français, ce qui arrive plus souvent qu’on ne le pense, mais généralement on ne l’apprends qu’après les avoir envoyé paître en français, ce qui n’apprécieront pas forcément.

Ah parce que je vous ai pas dit, en fait je viens d’Argentine, oui. Vous savez, là-bas le français c’est génial pour draguer. Ah, non, mais c’est énorme. Je ne sais pas pourquoi, mais cela explique pourquoi on dit que Buenos Aires c’est le “Paris de l’Amérique Latine”; en fait on a aussi un périf, des présidents corrompus, une équipe de foot qui a su gagner des titres il y a très longtemps, et une équipe de rugby qui promet d’aller très loin un jour.

Parce que comme je disais, je viens de l’Argentine. Un jour de 1987 je me rappelle que j’ai reçu une lettre de l’ambassade Suisse, et il y avait à l’intérieur un document qui disait en allemand, français et italien que j’étais suisse. Evidemment je ne l’ai pas su tout de suite, car je ne parlais ni allemand, ni français, ni italien; je suis donc allé au bar le plus proche pour appeler ma mère au travail - oui, on n’avait pas le téléphone. En fait, il y avait 5 ou 6 téléphones dans un rayon de 1000 mètres autour de la maison. Il devait y en avoir plus, mais les voisins qui en avaient un le gardaient en secret, parce qu’ils voulaient pas le prêter, ben oui vous vous imaginez bien qu’aider les gens qui

ont besoin d'une ambulance ou d'appeler la police ou les pompiers, ce n'est pas pratique, surtout lorsque cela arrive souvent.

Du coup j'ai fait les 500 mètres qui me séparaient du téléphone disponible le plus proche. C'était dans un bar, un vieux bar avec une grosse montre Longines accrochée au mur (quand je dit grosse, je veux dire du genre un mètre de diamètre) avec l'effigie de Carlos Gardel à côté. Ce bar se situait a l'intersection de Arenales et Azcuénaga, dans le quartier de Vicente Lopez. Evidemment cela ne vous dit rien, mais s'était aussi le terminus de la ligne 133, a l'époque ou elle s'arrêtait dans le bas Vicente Lopez, ce qui, comme vous vous en doutez certainement, ce n'est plus le cas, et en plus, cela n'a pas le moindre rapport avec ce que je vous raconte.

Donc je me suis rendu a ce bar, le vieil Espagnol qui le tenait me connaissait déjà. Dans le salon on entends un tango. On est bien en Argentine. Je le salue, je lui demande si on peut utiliser le téléphone, il me répond par l'affirmative. Je me dirige vers le téléphone en question, je décroche.

Pas de tonalité. Je le fais savoir a l'espagnol, qui me dit que depuis hier il n'y a pas de tonalité. Je me demande pourquoi il ne m'a pas dit cela avant de décrocher.

Miracle, quelques secondes plus tard, alors que j'étais a point de raccrocher, la tonalité se fait entendre. Je lui dit qu'il y a tout à coup la tonalité. Il me dit que oui, qu'après quelques minutes la tonalité apparaît toute seule. Je me demande pourquoi il ne me l'a pas dit avant.

Bref, j'appelle ma mère (il faut se dépêcher, il pourrait y avoir d'autres surprises imprévues et pas encore révélées par l'espagnol). Je raconte a ma mère a propos de la lettre. Elle me demande si j'ai mangé à midi. Je lui dit que oui et qu'il y a une lettre de l'ambassade. J'entends qu'elle parle a une collègue de travail. Elle me demande si ça a été ce matin au collège. Je lui dit que oui et je pense, en secret, que tout cela m'agace un peu. Cela fait 5 minutes que j'essaie de savoir ce que cette foutue lettre de l'ambassade veut dire et ni l'espagnol ni ma mère semblent s'en soucier. Bref.

Ma mère réagit ensuite a l'histoire de la lettre de l'ambassade, et me demande tout naturellement "ah oui! Et qu'est-ce que dit la lettre?". Ma réponse est simple: "Je ne sais pas, c'est en allemand, français et italien". Il faut dire que bien que je prenais des cours de français après le collège, mon niveau n'était pas à la hauteur d'une lettre officielle de l'ambassade, loin de là. Tout juste si je pouvais demander un café au lait avec des croissants au monsieur Dupont à Paris. Et Paris ce n'est pas en Suisse, alors c'est mal parti, parce qu'ici c'est plutôt un renversé, ou même un schale mit gipfeli bitte, alors tu vois de quoi ça m'a servi d'apprendre le français au collège a Buenos Aires.

Comme ma mère était très contente d'avoir reçu la lettre, je la lui lit; comprenons-nous bien; je la lui lit en espagnol; je lis du français

comme si s’était du castillan. Je vous laisse le soin d’imaginer le résultat d’une telle opération.

Je pense que ma mère a eu pitié de moi et n’a pas trop rigolé. Mais elle avait compris le fond de la question: j’avais reçu la nationalité suisse.

(à suivre)

Dropping support for iOS 5 and older

Adrian Kosmaczewski

2012-09-30

Just like last year[1], this year too we announce that all of our future projects will only be compatible with iOS 6 from now on.

That's right, we are dropping support of iOS 5, following the exponential rise of adoption[2] of the latest version, and also to take advantage of many new features only available in this new release.

Stay tuned for more updates of our products!

[1] /blog/dropping-support-for-iphone-os-3.x/

[2] http://www.apple.com/pr/library/2012/09/24iPhone-5-First-Weekend-Sales-Top-Five-Million.html

Hay un Lenguaje llamado COBOL

Adrian Kosmaczewski

2012-11-30

Soy un escritor. Reivindico mi pertenencia a un subconjunto de la raza humana que escribe. Pero no soy un escritor típico, porque en regla general no son los seres humanos los que leen lo que escribo. Sino maquinas.

Dicho asi suena como si yo fuese un mago. Y en el siglo 21, es un poco asi: escribo programas para computadoras, tambien conocidos en la jerga como “software”.

Literalmente, la palabra “software” significa “lo blando”, o “cosa blanda”. La palabra la inventaron los yanquis, en contraposicion o yuxtaposicion a lo que ellos denominan “hardware”. Esta ultima palabra ya se usaba desde la revolucion industrial, probablemente desde antes, para denominar la chatarra, lo ferreo, lo metalico, las maquinas como la de tejer o el tren.

Los simbolos de las dos primeras revoluciones industriales, las de Hobsbawm, eran llamadas comunmente “hardware”.

En la tercer revolucion industrial, aparecio el “software”, lo intangible, lo etereo, y lo pelotudo que me siento tratando de explicar esto usando palabras tan idiotas. Sigamos.

He aqui la verdad, la que mi abuela nunca supo (por razones que seran obvias, espero): un programa se escribe, asi como un escritor escribe su novela. En ambos casos se usa un teclado (aunque dudo que Hemingway hubiese escrito programas con su Remington del ’32), y el resultado final se lee de izquierda a derecha, en lineas que se suceden de arriba para abajo.

Siempre me pregunte lo que seria la programacion si los arabes no hubiesen detenido sus investigaciones matematicas, por ejemplo despues de inventar los numeros arabigos, o despues de publicar aquel best seller de Al-Khowarizmi en el siglo 11. Escribiríamos los programas de derecha a izquierda?

Lo que quedo del buen señor Al-Khowarizmi fue su nombre, que derivo en la palabra “algoritmo”, que es una palabra complicada para designar recetas de cocina.

Decia, entonces, que escribir un programa es como escribir un libro. Claro que no es usted quien lee el libro en cuestión. En realidad usted

hace doble clic en un dibujito en su pantalla y no ve lo que yo he escrito; usted “ejecuta” el programa, y en realidad, en su pantalla usted ve una pelicula.

Porque en realidad, lo que uno escribe como programador se parece mas a guiones de cine que a otra cosa. Hay personajes, accion, suspenso, drama, persecuciones y emociones, todas sucediendo entre usted y lo que ve en la pantalla.

Pantalla que le devuelve una version edulcorada de lo que sucede, en realidad, en el interior de su computadora. Y no me venga con las imagenes de “Tron” o “Matrix”, porque en realidad, la cosa es mucho mas violenta, como una mezcla de “Full Metal Jacket” con “Dumb and Dumber” y “When Harry Meets Sally”. Y una pizca de “Madagascar”, sobre todo la parte con los pinguinos milicos. No le miento.

Es tan violenta que hasta nosotros mismos, los programadores, usamos lenguajes que no son los de la computadora. Porque la computadora necesita que le traduzcamos lo que le decimos; imaginese que el lenguaje de la maquina es tan intricado que solo algunos iniciados lo conocen. Por ahi los ha visto usted en algun documental sobre el tema; unos barbudos californianos convencidos de que el mundo empezó el 1ro de enero de 1970.

En la jerga, al proceso de traduccion le decimos “compilacion”, porque es como juntar muchos exitos de Eddy Mitchell en un disco y no morir en el intento. Si no saben quien es Eddy Mitchell, no saben que suerte tienen. Basicamente, al compilar el programa escrito por nuestras manos llenas de sudor, cafe y caspa, se obtiene un “algo” dificil de definir, pero que aparentemente se asemeja a una serie de unos y ceros, y que la computadora comprende lo suficientemente bien como para fallar al primer error que encuentra, y asi mostrarnos una ventanita de “dialogo” (que palabra mas optimista, cuando lo unico que se le puede contestar es “ok”, “cancel”, “retry” o “abort”), que es la version computadorizada de mandarnos a freir churros, o, al menos, a preparar otra taza de cafe antes de empezar todo de vuelta.

Y al lenguaje propio de la maquina, le decimos “assembler” o “ensablador”, porque son los ladrillos ultimos con los que se ensamblan estos juegos de Lego diabolicos en los que nos encontramos.

A los lenguajes que se “compilan”, se les pone unos nombres muy originales, y la verdad es que hay centenas, miles de lenguajes distintos. Si usted piensa que aprender idiomas es tedioso, gran parte de nuestra vida y de nuestro trabajo de programador consiste en, justamente, aprender nuevos idiomas todo el tiempo.

Pero ojo, que no es solamente el idioma, sino sus dialectos locales, con sus “patois” y sus expresiones tipicas de cada region.

Igual, no nos quejemos tanto, ya que no son idiomas tan complejos como los humanos: en regla general un lenguaje de programacion no

cuenta mas de 40 o 50 palabras, y tienen, en regla general, estructuras gramaticales que se asemejan muchisimo entre si; aprender un nuevo lenguaje de programacion despues de que se aprendio el primero toma, en promedio, un par de semanas, y obviamente, la practica ayuda a reducir esos tiempos.

Los lenguajes de programacion se especializan: los hay para todos los gustos y colores. Algunos son buenos para calculo estadistico, otros son mejores para dibujar. Algunos son ideales para mandar cohetes al espacio, otros sirven para conectarse a Facebook o Twitter.

Y hay una gran mayoria que pueden, con mayor o menor dificultad, ser usados para cualquier cosa.

Por ejemplo, hay un lenguaje que se llama COBOL; el nombre es una sigla que significa literalmente “lenguaje comun orientado a negocios”. Tiene varias particularidades que lo hacen indicado para trabajar en un ambitos bancarios: es verborragico a muerte y SE ESCRIBE TODO EN MAYUSCULAS. Por ejemplo, en COBOL se le dice a la compu

```
ATENCION ACA ARRANCA PROGRAMA
CARGAME EL CODIGO EN MEMORIA PARA ESCRIBIR TEXTO
ABRI CONEXION CON LA BASE DE DATOS
VOY A ESCRIBIR ALGO ATENTI CATALINA
ESCRIBI EN LA PANTALLA "QUE HACELGA"
AHORA EL PROGRAMA ESTA POR TERMINAR
AVISEN A ADMINISTRACION QUE YA NOS VAMOS
UN SALUDO A MI VIEJA QUE ME ESTA MIRANDO.
```

Otros lenguajes son menos vistosos. Por ejemplo, en Assembler el mismo programa es mucho mas dificil de escribir, y basicamente consiste en decirle al chip Pentium que tenes en la compu que haga lo mismo, pero mas complicado:

```
prep memori
guard "que hacelga"
ponr texto, memoria
prep pantalla
escr variable
tfuiste
```

Los lenguajes de programacion llevan puesta la neurosis de su creador. Cuanto mas esquizofrenico el inventor, mas estravagante el lenguaje. Y la esquizofrenia es moneda corriente en nuestra industria. Creanme, estar muchas horas delante de una computadora no es inocuo.

Por ejemplo, uno de los casos emblematicos de lenguaje esquizofrenico es uno llamado “Python”, que fue creado por un holandes que era fanatico de los Monty Python. La consigna de base del lenguaje es que “existe una sola manera de hacer las cosas correctamente” (no es broma), la cual, obviamente, parte del punto de vista miope e irracional de su autor. No voy a dar un ejemplo de este lenguaje porque es tan aberrante como popular. Pareciera que

a muchos programadores les encanta dejar de pensar; la verdad es que, conociendo a muchos fanaticos de Python, no me extrana.

Obviamente, al decir esto me expongo a insultos, amenazas de muerte, reuniones del Ku Klux Klan delante de mi casa, o peor aun, autos con megafonos tocando temas de Pimpinela presentados por Mateyko por doquier.

Porque de la misma manera que los lenguajes de programacion llevan en si el germen de la psicosis de su creador, tambien aglutinan a su alrededor a verdaderas “tribus”, cuyos enfrentamientos hacen que un River-Boca se asemeje a una discusion de jubilados en una cancha de bochas en algun pueblo del Tirol.

20 Moments of Argie Paleomarketing

Adrian Kosmaczewski

2013-01-09

For the past 4 years I've been sharing old argentine advertising campaigns on Twitter, with the meme **"Moment of argie paleomarketing brought to you by akosma™®©"**. The jingles and slogans (mostly from the 80's) would pop up in my head at any random moment, and would always be followed by chuckles, sarcasm, puzzled looks and sometimes even hatred from my argie followers.

But I realized that I've done that 20 times since 2009! To celebrate this milestone (?) here goes the complete list with the links to the original tweets. Enjoy!

Yoooo coltal tolta![1]

Dánica dorada, Dánica dorada, era para untar, era para untar![2]

Coma manzaaana![3]

Neeeegro negro negro, blaaaanco blanco blanco![4]

A condensed moment of argie paleomarketing[5] on YouTube[6]!

Arrolla la seeed... Paso de los Torooos[7]

Koh-I-Noor: poderoso el chiquitín![8]

Que pan dulce, Pamela![9]

You know when you've been tango'ed[10]

Bardahl, bardahl, bardahl....[11]

Zeta... nnnoventaicinco[12]

[1] http://twitter.com/akosma/status/288746071784566784
[2] https://twitter.com/akosma/status/288342467974156290
[3] https://twitter.com/akosma/status/275454788316065792
[4] https://twitter.com/akosma/status/266480792102924288
[5] https://twitter.com/akosma/status/25622412660
[6] http://www.youtube.com/watch?v=4So1n4qgU9c&feature=related
[7] https://twitter.com/akosma/status/24099117076
[8] https://twitter.com/akosma/status/10361371266
[9] https://twitter.com/akosma/status/7012656536
[10] https://twitter.com/akosma/status/6673611575
[11] https://twitter.com/akosma/status/5763782867
[12] https://twitter.com/akosma/status/4446628397

Hitachi, que bien se T.V.[13]

Piense Fuerte. Piense Ford.[14]

Grundig: caro, pero el mejor[15]

Si es Bayer, es bueno[16]

Arroz Gallo: el de la caja, amariiiiiilla![17]

Cepita, la sana costumbre, Cepitaaaa![18]

Como estoy?!?!? Uhhh!!![19]

Shampoo Valet, para la caspa: esto es verdad[20]

Pan, pan pan, pan pan es de Sacaan / pan, pan pan, Sacaan es el buen pan[21]

And one more, from my Instagram[22] this time:

[13] https://twitter.com/akosma/status/3951179744
[14] https://twitter.com/akosma/status/3631623677
[15] https://twitter.com/akosma/status/3401351345
[16] https://twitter.com/akosma/status/3100898780
[17] https://twitter.com/akosma/status/2877037895
[18] https://twitter.com/akosma/status/2685782590
[19] https://twitter.com/akosma/status/2601533837
[20] https://twitter.com/akosma/status/2587087918
[21] https://twitter.com/akosma/status/2584659938
[22] http://instagram.com/p/MQwVXfFU5q/

[23]

[23] http://instagram.com/p/MQwVXfFU5q/

Best Books of 2012

Adrian Kosmaczewski

2013-01-18

Ahhh... it is that time of the year again, just like in 2011[1], 2010[2], 2009[3], 2008[4] and 2007[5].

And in this particular occasion, not only I can write about the books I've read, but also about the one I've written. But let's start with the books I've read: in the technical category, these are the ones I've enjoyed in 2012 (in no particular order):

- What is Node?[6], by Brett McLaughlin
- Test Driving iOS Development with Kiwi[7], by Daniel Steinberg
- Test-Driven iOS Development[8], by Graham Lee
- Publishing with iBooks Author[9], by Nellie McKesson, Adam Witwer
- tmux: Productive Mouse-Free Development[10], by Brian P. Hogan
- Programming in Lua, Second Edition[11], by Roberto Ierusalimschy

In the non-technical category:

- Inside Apple: How America's Most Admired–and Secretive–Company Really Works[12], by Adam Lashinsky
- Griftopia[13], by Matt Taibbi
- The Elements of Style[14], by William Strunk Jr.
- Einstein[15], by Walter Isaacson

[1] /blog/best-books-of-2011/
[2] /blog/best-books-of-2010/
[3] /blog/best-books-of-2009/
[4] /blog/best-books-of-2008/
[5] /blog/best-books-of-2007/
[6] http://shop.oreilly.com/product/0636920021506.do
[7] http://editorscut.com/Books/001kiwi/001kiwi-details.html
[8] http://www.amazon.com/Test-Driven-iOS-Development-Developers-Library/dp/0321774183
[9] http://shop.oreilly.com/product/0636920025597.do
[10] http://pragprog.com/book/bhtmux/tmux
[11] http://www.inf.puc-rio.br/~roberto/pil2/
[12] http://www.amazon.com/Inside-Apple-Americas-Admired-Secretive-Company/dp/145551215X
[13] http://www.amazon.com/Griftopia-Bankers-Politicians-Audacious-American/dp/0385529961
[14] http://www.amazon.com/Elements-Style-William-Strunk-Jr/dp/1557427283
[15] http://www.amazon.com/Einstein-ebook/dp/B000PC0S0K/ref=tmm_kin_title_0

- What I Talk About When I Talk About Running[16], by Haruki Murakami
- Graphic Design: The New Basics[17], by Ellen Lupton and Jennifer Cole Phillips

Finally, I'm very proud to say that O'Reilly published my first book, Mobile JavaScript Application Development[18], in print and electronic formats (PDF, ePub and Kindle), and received excellent reviews![19]

PS: I say this is the first... because the second will be published very soon :) stay tuned for more news.

[16] http://www.amazon.com/What-Talk-About-Running-ebook/dp/B005TKD8ZK/ref=tmm_kin_title_0

[17] http://gdbasics.com

[18] /blog/announcing-my-first-book-mobile-javascript-application-development/

[19] /blog/praise-for-mobile-javascript-application-development/

Retrospective 2012

Adrian Kosmaczewski

2013-01-24

Just like in 2010[1] and 2011[2], here goes a small recapitulation of the main events that happened in akosma software during 2012; another great year with incredible achievements, and a bright future ahead.

I would like to start by thanking all of my partners, clients, friends and family, plus all of my followers in Twitter[3], Facebook[4], Github[5] and LinkedIn[6], who have been extremely supportive to help me achieve incredible milestones this year. I could have certainly not have done it without you guys! Thanks a lot to all of you (you know who you are!)

The Book

[1] /blog/retrospective-2010/
[2] /blog/retrospective-2011/
[3] http://twitter.com/akosmasoftware
[4] https://www.facebook.com/akosmasoftware
[5] https://github.com/akosmasoftware
[6] http://www.linkedin.com/company/akosma-software

By far, the event that did myself the proudest last year was finally publishing my first book, “Mobile JavaScript Application Development”[7] at O’Reilly. This book is an evolution of the booklet written for our students of the first mobile web training in Zürich in 2011[8].

The book had outstandingly positive reviews so far[9] and I’d like to thank my friends of immedia[10] in South Africa for hosting a very nice launch party[11] in July!

I also say “my first book” because, well, the second one is in production and will be hopefully published very soon!

Training

In 2012 I have given a total of 336 hours of training, to 88 students in Zürich, Paris, Cape Town, Johannesburg and Durban! I have taught iOS (both beginner and advanced levels) and mobile web development (a subject particularly interesting for enterprise developers).

[12]

We have dramatically extended the collaboration with our friends of immedia[13], which yielded 4 sessions of mobile application training in South Africa[14] during February, July, October and December. I am proud and happy to have been able to empower so many developers to embrace the technologies required for this new post-PC world ahead of us!

There was also another edition of the successful Mobile Web Training in Zürich in March[15], and last but not least, I have also collaborated

[7] http://shop.oreilly.com/product/0636920025252.do
[8] /blog/mobile-development-with-jquery-sencha-and-phonegap-15-17-november-2011/
[9] /blog/praise-for-mobile-javascript-application-development/
[10] http://www.immedia.co.za
[11] /blog/mobile-javascript-application-development-book-launch-event-in-south-africa/
[12] http://www.immedia.co.za
[13] http://www.immedia.co.za
[14] http://www.appdev.co.za
[15] /blog/mobile-web-training-in-zurich/

with Sencha[16] during the month of November, to give the official Sencha Touch training curriculum[17] in Paris.

Conferences

Last year I had the privilege of organizing and hosting the mobile track at QCon London in March[18]; we had a great lineup of speakers including Maximiliano Firtman, Tobie Langel and Christophe Coenraets.

As a speaker I spoke about Sencha Touch twice, in JavaScript events in Switzerland; first in March, during the official JS Genève meetup[19], and then in June, during SwissJeese 2012 in Bern[20].

Finally, I had the opportunity to attend Mobilism in May[21] and WWDC in June[22].

Apps

This year I have helped my clients publish new apps:

- The Henri Dès application[23], and
- The iOS version of the EERV Lausanne Cathedral Applications[24] in English, French and German versions.

[16] http://www.sencha.com
[17] http://www.sencha.com/training
[18] /blog/this-week-qcon-london-2012/
[19] /blog/tonight-in-geneva-introduction-to-sencha-touch-2/
[20] /blog/talking-at-swissjeese-2012/
[21] /blog/this-week-mobilism-2012-amsterdam/
[22] /blog/attending-wwdc-2012/
[23] /blog/introducing-the-henri-des-iphone-app-la-radio-hd/
[24] /blog/introducing-the-lausanne-cathedral-iphone-applications/

Talking about the Lausanne Cathedral apps, I would like to mention that the Android apps were developed by the incredible Android team of immedia[25], led by Duncan Scholtz[26]!

And more!

I have also had the privilege of being part of the Jury of the first edition of the Swiss App Awards[27], held in Zürich in March. Finally, I've also helped immedia[28] during the month of August as a technical advisor to the management and technical teams.

PS: by the way, the second edition of the Swiss App Awards has just announced the shortlist of candidates![29]

[25] http://www.immedia.co.za
[26] https://twitter.com/duncandee2
[27] /blog/swiss-app-awards-2012-winners/
[28] http://www.immedia.co.za
[29] http://swissappawards.ch/swiss-app-awards-2013/shortlist/

Depresión

Adrian Kosmaczewski

2013-01-29

Es algo que me sucede relativamente seguido.

Es algo que nunca le conté a la vieja, tal vez para no preocuparla. Pero es algo que me costó varias relaciones, canas, arrugas, lágrimas, y que me sucede de maneras erráticas.

En mi caso, las crisis se desencadenan de maneras improvistas.

Y cuando sucede, mi mundo se derrumba. Todo se viene abajo. Nada tiene salvación, y en lo que me respecta, siento que el planeta entero podria explotar en el minuto que sigue, que no me importaría.

En estas situaciones, no es que estoy triste o eufórico; estoy simplemente perdido, sin emoción alguna. Un desgano terrible se apodera de mi.

Y tengo solo una cura conocida; crear. Las veces que salí mejor del asunto fueron cuando decidí empezar a escribir un libro nuevo, a armar algo con Lego, a pensar en algun nuevo software, algo así. Tengo que empezar a pensar en algo distinto, en sacar de adentro mío algo nuevo. No entiendo bien como sucede esto, pero al menos hasta ahora, funciona.

Pido disculpas a quienes me rodean en esos momentos. Es como si no fuese yo. En particular, le pido perdón a Claudia.

Es como si hubiese algo dentro de mí que pide salir, a gritos, pero no sé que es, o como dejarlo irse.

Announcing my second book: “Sencha Touch 2 Up and Running”

Adrian Kosmaczewski

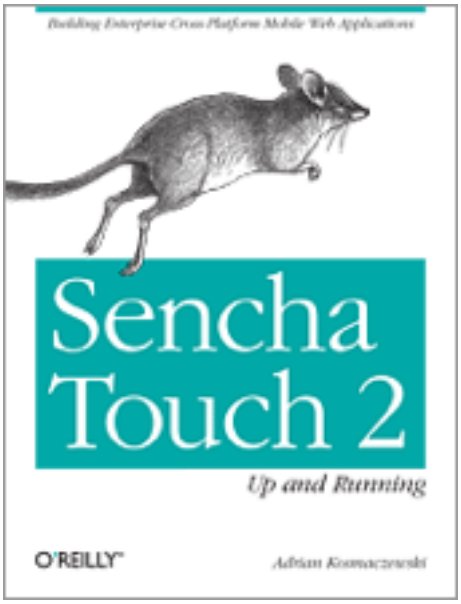
2013-02-26

This is the official announcement of my second book, “Sencha Touch 2 Up and Running”[1], available for sale in print and in DRM-free electronic formats (PDF, ePub, Kindle, etc.). This book is my humble attempt to provide an easy path to learn the basics of Sencha Touch 2.1.

This is the book description:

> Launch into Sencha Touch 2 with this hands-on book, and quickly learn how to develop robust mobile web apps that look and behave like native applications. Using numerous code samples, author Adrian Kosmaczewski guides you every step of the way through this touchscreen-enabled JavaScript framework—from creating your first basic app to debugging, testing, and deploying a finished product.
>
> Learn how to craft user interfaces, build forms, and manage data, then deploy as either an HTML5 offline app or as a native app for Android, iOS, or Blackberry.

[1] http://shop.oreilly.com/product/0636920026877.do



You can buy a copy of the book at these locations:

- O'Reilly[2]
- O'Reilly Safari Books Online[3]
- iBooks Store[4]
- Amazon.com[5]
- Amazon.de[6]
- Amazon.ca[7]
- Amazon.co.uk[8]
- Amazon.es[9]
- Amazon.it[10]
- Amazon.fr[11]
- Amazon.co.jp[12]
- Amazon.cn[13]
- Barnes and Noble[14]
- Thalia.ch[15]

[2] http://shop.oreilly.com/product/0636920026877.do

[3] http://my.safaribooksonline.com/book/programming/javascript/9781449339371

[4] https://itunes.apple.com/us/book/sencha-touch-2-up-and-running/id610869809?mt=11

[5] http://www.amazon.com/Sencha-Touch-2-Up-Running/dp/1449339387

[6] http://www.amazon.de/Sencha-Touch-2-Up-Running/dp/1449339387

[7] http://www.amazon.ca/Sencha-Touch-2-Up-Running/dp/1449339387

[8] http://www.amazon.co.uk/Sencha-Touch-2-Up-Running/dp/1449339387

[9] http://www.amazon.es/Sencha-Touch-2-Up-Running/dp/1449339387

[10] http://www.amazon.it/Sencha-Touch-2-Up-Running/dp/1449339387

[11] http://www.amazon.fr/Sencha-Touch-2-Up-Running/dp/1449339387

[12] http://www.amazon.co.jp/Sencha-Touch-2-Up-Running/dp/1449339387

[13] http://www.amazon.cn/Sencha-Touch-2-Up-Running/dp/1449339387

[14] http://www.barnesandnoble.com/w/sencha-touch-2-up-and-running-adrian-kosmaczewski/1114198364

[15] http://www.thalia.ch/shop/ebook-e-book-ebooks-e-books-4893/suchartikel/sench

- Lehmanns.ch[16]

This second book would have never been possible without the great help of lots of incredible people scattered all over the planet; to begin with, the whole Sencha[17] team, who have created and documented an out-of-this-planet kind of JavaScript framework; kudos and thanks to all of them, in particular to Jeff Hartley, Vice President of Services; to David Marsland, Chief Instructor; and Jim Soper, Senior Technical Trainer at Sencha.

I would also like to thank Simon St. Laurent, my editor at O'Reilly, who wholeheartedly embraced the idea of this book just as we were sending "Mobile JavaScript Application Development"[18] to press, and was extremely supportive during the process. I would also like to thank the reviewers of this book. Jens-Christian Fischer, from Zurich, Switzerland, with whom I had the privilege of teaching Sencha Touch in the past, and who has provided me with incredible tips and tricks to make this book a better one. Mats Bryntse, Founder of Bryntum AB from Lund, Sweden, creator of the Siesta testing framework described in Chapter 8, and who reviewed it extensively. Gabriel García Marengo, web designer at IMD in Lausanne, Switzerland, who sent great feedback, and who is one of the best friends anyone could have. Martín Paoletta, Solutions Architect at Redbee in Buenos Aires, Argentina, who read the book from the perspective of a solution provider, and made excellent recommendations. Thanks to you all.

This book is dedicated with love and gratitude to Claudia. Te amo, preciosa.

The code samples of the book can be found, as usual, in Github[19]. Feel free to fork, share and enjoy! I hope this book will be useful to those developers looking for a good introduction to the subject.

---

a_touch_2_up_and_running/adrian_kosmaczewski/ISBN1-4493-3938-7/ID34202485.html

[16] http://www.lehmanns.ch/shop/mathematik-informatik/25504905-9781449339388-sencha-touch-2-up-and-running

[17] http://www.sencha.com/

[18] http://shop.oreilly.com/product/0636920025252.do

[19] https://github.com/akosma/Sencha-Touch-2-Up-And-Running

New Swiss Training Calendar 2012

Adrian Kosmaczewski

2013-04-08

We are very glad to announce new dates for our Post PC trainings in Switzerland! The new website training.akosma.ch[1] contains all the information, calendar, dates and prices for our offerings. We provide focused, practical training of the latest technologies, with more than 5 years of experience.

- iOS Beginners[2] (2 days, 1400 CHF): 15-17 April
- iOS Advanced[3] (3 days, 2100 CHF): 17-19 April
- Android Beginners[4] (2 days, 1400 CHF): 22-23 April
- Mobile Web App Development[5] (3 days, 2100 CHF): 24-26 April
- Mobile App Testing[6] (2 days, 1400 CHF): 29-30 April
- Sencha Touch Beginners[7] (3 days, 2100 CHF): 1-3 May

All prices include VAT / TVA / MwSt. The complete course calendar is available at training.akosma.ch[8]. Sign up now!

[1] http://training.akosma.ch
[2] http://training.akosma.ch/ios-beginners
[3] http://training.akosma.ch/courses/ios-advanced
[4] http://training.akosma.ch/courses/android-beginners
[5] http://training.akosma.ch/courses/mobile-web-app-development
[6] http://training.akosma.ch/courses/mobile-app-testing
[7] http://training.akosma.ch/courses/sencha-touch-beginners
[8] http://training.akosma.ch

PS: Feel free to contact us for customized training sessions in your own company[9]!

[9] http://training.akosma.ch/custom-trainings/

New Swiss Conference: GOTO Zürich!

Adrian Kosmaczewski

2013-04-10

We are proud to participate in the first GOTO Zürich for Developers[1] + GOTO Zürich For Leaders[2] Conferences as one of the sponsors[3]! The GOTO Conferences[4] happen all over the world, in Berlin, Amsterdam, Århus, Chicago, Prague and now Zürich too!

[5]

Join us today and tomorrow in the Marriott Zürich for two days featuring talks by great speakers such as Alvaro Videla[6], Bruce Tate[7], Beat Schwegler[8], Dan North[9], Janne Jul Jensen[10], Jim Webber[11], Michele Ide Smith[12] and Scott Ambler[13].

Two tracks will feature content about mobile applications and we strongly recommend you check them out!

[14]

- Mobile & Usability, Interaction Design[15]

[1] http://gotocon.com/zurich-2013/
[2] http://gotocon.com/zurich-leaders-2013
[3] http://gotocon.com/zurich-2013/sponsors/
[4] http://gotocon.com/
[5] http://gotocon.com/
[6] http://gotocon.com/zurich-2013/speaker/Alvaro+Videla
[7] http://gotocon.com/zurich-2013/speaker/Bruce+Tate
[8] http://gotocon.com/zurich-2013/speaker/Beat+Schwegler
[9] http://gotocon.com/zurich-2013/speaker/Dan+North
[10] http://gotocon.com/zurich-2013/speaker/Janne+Jul+Jensen
[11] http://gotocon.com/zurich-leaders-2013/speaker/Jim+Webber
[12] http://gotocon.com/zurich-2013/speaker/Michele+Ide-Smith
[13] http://gotocon.com/zurich-2013/speaker/Scott+Ambler
[14] http://gotocon.com/zurich-2013
[15] http://gotocon.com/zurich-2013/tracks/show_track.jsp?trackOID=716

- Deciding Mobile[16]

[17]

Other interesting tracks include Good Code[18], Databases and NoSQL[19], Databases for the Cloud[20], Polyglot Programming[21], Lean Development[22] and Cool Companies[23]. See you there!

[16] http://gotocon.com/zurich-2013/tracks/show_track.jsp?trackOID=708
[17] http://gotocon.com/zurich-leaders-2013
[18] http://gotocon.com/zurich-2013/tracks/show_track.jsp?trackOID=729
[19] http://gotocon.com/zurich-2013/tracks/show_track.jsp?trackOID=710
[20] http://gotocon.com/zurich-2013/tracks/show_track.jsp?trackOID=737
[21] http://gotocon.com/zurich-2013/tracks/show_track.jsp?trackOID=736
[22] http://gotocon.com/zurich-2013/tracks/show_track.jsp?trackOID=807
[23] http://gotocon.com/zurich-2013/tracks/show_track.jsp?trackOID=728

Mysterious Ways (written while flying)

Adrian Kosmaczewski

2013-04-11

At that point Virgin Mary stood up and said, quite proudly:

“No way. The most beautiful song ever written to me was ‘Mysterious Ways.’ ”

“U2?” he asked, rather incredulous.

“Exactly. All the rest, you can forget.”

Bad, bad flight.

Adrian Kosmaczewski

2013-04-17

It’s bad to be stuck on a plane next to a person who snores with a wide open mouth. It’s even worse if the person in question is a massive rugby player and you’re stuck against the window. And even worse if he has bad breath. Add a child crying in the vicinity and you are seriously in trouble. Oh, and if the flight is more than 10 hours long, well, let’s just say it wasn’t your day. Like, at all.

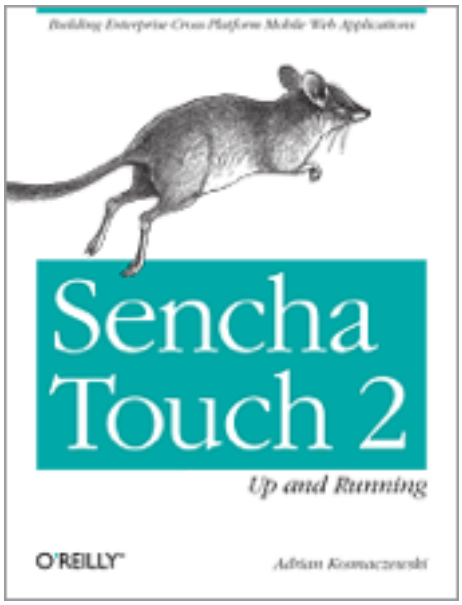
Praise for “Sencha Touch 2 Up and Running”

Adrian Kosmaczewski

2013-05-05

jasonou[1]:

> A very good book on Sencha Touch. This book covers various important topics. Learned many very useful information from it. Enjoyed it. (…)



Léo Stringaro[2]:

> Parfait pour bien débuter avec Sencha Touch. Permet de mettre en place, rapidement, une app complete. Explications claires. (…)

Anthony Emery[3]:

[1] http://www.amazon.com/review/R168YID4IMXYTC/ref=cm_cr_pr_perm?ie=UTF8&ASIN=B00BFZTI2U&linkCode=&nodeID=&tag=

[2] http://www.amazon.fr/review/R8APEW59J7HGA/ref=cm_cr_pr_perm?ie=UTF8&ASIN=1449339387&linkCode=&nodeID=&tag=

[3] http://www.amazon.co.uk/review/R2BKJD5IJIQM1Y/ref=cm_cr_pr_perm?ie=UTF8&ASIN=1449339387&linkCode=&nodeID=&tag=

> (…) The book does not attempt to lead you through the creation of a whole sample application allowing it to be concise whilst still feeling like they have covered an awful lot. I think this style is perfectly suited to an intermediately skilled programmer where little time is wasted on the basics and you get into the real 'meat' of the language quickly.
>
> Needless to say I have been very impressed by Sencha Touch 2 (as a framework) and I would definitely recommend this book as a good entry point.

Ben Love[4]:

> (…) The author is clearly comfortable with the subject matter and could have easily come across "professor-like" in his instruction. Instead, his passion for the framework shines through in his writing and makes it easy to get excited about what you can do with Sencha Touch when you're done with the book. I wish more frameworks had a book like this accompanying them. (…)

K. Addaquay[5]

> i have read most of the recent sencha books and this one is definitely one of my favorite. i think the author has done a great job at simplifying the inner workings of sencha touch 2. (…)

Kostantin[6]:

> (…) I recently started sencha development and i have read all the new books on sencha touch 2… but somehow i really love this book … how its organized. (…) The examples given in this book are the best all all the books i have! And i found a wealth of information on chapter 3,4 and 6 and 7 (Views, Data, Controller and Styling Applications).. i felt the other books were not satisfactory. Other books were missing information on framework functions, utilities and component information, and this book provided them for me. i am definitely keeping this book as a ref guide (…)

You can buy a copy of the book at these locations:

- O'Reilly[7]
- O'Reilly Safari Books Online[8]
- iBooks Store[9]

[4] http://www.goodreads.com/review/show/547330932

[5] http://www.amazon.com/review/R1TFAI8DU3X1EP/ref=cm_cr_pr_perm?ie=UTF8&ASIN=1449339387&linkCode=&nodeID=&tag=

[6] http://shop.oreilly.com/product/0636920026877.do#PowerReview

[7] http://shop.oreilly.com/product/0636920026877.do

[8] http://my.safaribooksonline.com/book/programming/javascript/9781449339371

[9] https://itunes.apple.com/us/book/sencha-touch-2-up-and-running/id610869809?mt=11

- Amazon.com[10]
- Amazon.de[11]
- Amazon.ca[12]
- Amazon.co.uk[13]
- Amazon.es[14]
- Amazon.it[15]
- Amazon.fr[16]
- Amazon.co.jp[17]
- Amazon.cn[18]
- Barnes and Noble[19]
- Thalia.ch[20]
- Lehmanns.ch[21]

[10] http://www.amazon.com/Sencha-Touch-2-Up-Running/dp/1449339387
[11] http://www.amazon.de/Sencha-Touch-2-Up-Running/dp/1449339387
[12] http://www.amazon.ca/Sencha-Touch-2-Up-Running/dp/1449339387
[13] http://www.amazon.co.uk/Sencha-Touch-2-Up-Running/dp/1449339387
[14] http://www.amazon.es/Sencha-Touch-2-Up-Running/dp/1449339387
[15] http://www.amazon.it/Sencha-Touch-2-Up-Running/dp/1449339387
[16] http://www.amazon.fr/Sencha-Touch-2-Up-Running/dp/1449339387
[17] http://www.amazon.co.jp/Sencha-Touch-2-Up-Running/dp/1449339387
[18] http://www.amazon.cn/Sencha-Touch-2-Up-Running/dp/1449339387
[19] http://www.barnesandnoble.com/w/sencha-touch-2-up-and-running-adrian-kosmaczewski/1114198364
[20] http://www.thalia.ch/shop/ebook-e-book-ebooks-e-books-4893/suchartikel/sencha_touch_2_up_and_running/adrian_kosmaczewski/ISBN1-4493-3938-7/ID34202485.html
[21] http://www.lehmanns.ch/shop/mathematik-informatik/25504905-9781449339388-sencha-touch-2-up-and-running

Shutting Down

Adrian Kosmaczewski

2013-05-22

After 5 years, akosma is shutting down its operations completely.

We would like to thank all of you, friends, family, customers, partners, who have helped us achieve incredible milestones and projects. We have participated in incredible projects, traveled around the world and done many things that, when looking back, seemed simply impossible when we started.

What's next? No idea, but we'll let you know for sure. We are a bit sad, but at the same time this opens up lots of doors for new projects and adventures.

Thanks again, and see you soon!

Adrian

akosma software

Adrian Kosmaczewski

2013-05-23

They say things come in pairs in this life.

Yesterday I closed my company, akosma software, also known by the Swiss government as “akosma software - Adrian Kosmaczewski”, federal number CH-550-1058663-5. This company was born out of a whim, a desire for change. I had toyed with the idea of starting my own business for years, without fully realizing the insane amount of work and dispair that it brings, together with an almost boundless freedom of creation.

As a matter of fact, akosma software was literally a byproduct of the App Store and the Post-PC world. When the iPhone first came out, in 2007, I told my wife that should Apple publish an SDK for it, we were going to San Francisco, to attend WWDC and right after to start a business around it. And so it happened, when in March 2008 they released the first iteration of what would later be known as the iOS SDK.

June 2008 saw me among those who bought a ticket to see el Jobso introduce the iPhone 3G, version 2.0 of the iPhone OS, and, well, MobileMe. Anyway. I came back to Switzerland to start writing iPhone apps, much to the skepticism of almost everyone around me. It took me 1 year and a half, maybe even 2 and the arrival of the iPad, to actually be able to make a decent living out of akosma software. The local market took some time to take off, but when it did, it literally exploded.

As an anecdote, it is interesting to remember that I got fired from my job a week after returning from WWDC 2008, and I was somehow forced to start my business right away.

Another intesting anecdote was meeting Daniel Pasco during that first WWDC: hearing him urging me to start my own business around the iPhone was a big influence to me.

The most profitable run of akosma software lasted probably 2, maybe 2 years and a half. During that time I turned down investment and buyout proposals; I truly wanted to go through the creation of a one-man company all by myself, to see what was behind it. I wanted to grow akosma organically. I did not want to do any offshoring or outsourcing. I almost hired someone, as a matter of fact. I taught

myself taxes, financial plans, NDAs, contracts, negotiation, invoicing, basic accounting, all while dealing with bugs, breakpoints, IDEs, autorelease pools and memory leaks.

I attended my fourth WWDC last year, knowing that it would be my last; not because the soldout times have become insane (which acts as a natural barrier to attending, anyway) but because the whole thing was over for me. After Objective-C had been named the most popular programming language two years in a row by TIOBE; after half a billion iOS devices had been sold; after countless new companies started doing an incredible job at mobile app making, the appeal of the novelty, the thrill of the discovery was over. iOS will grow and still be relevant for the next 10 years (at least). I will probably still work in this industry for some time. But not like this.

What happened, then? In many ways, akosma software can be both seen as a failure and as a success.

If I say that I failed, of course, there is nobody to blame but me. Last year, as I felt that the market was changing, as bigger and better companies were doing what I was doing, I concentrated my efforts in the one segment that brought me the most happiness, and which I thought would grow and foster the most: teaching. However, that alone could not sustain my family, so I decided after a year of trying different formulí¦, that it was time to throw the towel.

On the other side, I also think that akosma software was my biggest success to date. As a last wish, I hope that it made a difference. I hope that the code left behind in Github, the answers in Stack Overflow, the blog posts, the books, and the teaching lessons will live on. Yesterday one of my first South African students left me a message in my Facebook page saying “thanks for teaching me iOS”. I take that as the legacy of akosma software. I take that as a success.

What is next? I actually do not know. I am currently looking for job options, evaluating quite a few interesting ones, and maybe one day I will own a business again, who knows. I learnt quite a few things about making a business in Switzerland, so I hope to use that knowledge again in the future.

I want, again, to thank all of you; your support humbles me and I am very glad that you were there, paying attention and giving me feedback in every step of the way. In these 5 years I have learnt more that I would have ever thought I could. I met incredible people, in 4 different continents. I have spoken at many conferences. I have had hundreds of students. I have even reached premium frequent flyer status at a couple of airlines. I guess I will use those miles for some time off soon. I could use some time in a white, sunny beach, somewhere.

In no particular order, I want to name some key people I have met or worked with during these five years, all of whom have been fundamental to akosma software in many different ways: Daniel

Steinberg, Mike Lee, Jí¸rn Larsen, Anice Hassim, Gabriel García Marengo, Dominique Jost, Daniel Pasco, Ciro Mondueri, Graham Lee, MC Casal, Sabine Dufaux, Daniel Fozzatti, Jens-Christian Fischer, Selene Shah, Karen Barber, Danilo Campos, Ela Kosmaczewska, Paul Buck, Florent Pillet, Fabien Kupferschmid, Joe D'Andrea, Cédric Lüthi, Raven Zachary, Barnaby Skinner, Erica Sadun, Erasmo de Falco, Simon St.Laurent, Jonas Schnelli, Devaprakash Giretheren, Maximiliano Firtman, Victoria Marchand, Hernán Pelassini, Junior Bontognali, Tobie Langel, Bertrand Dufresne, Patrick Chareyre, Thierry Weber, Yannis Jacquet, Carlos Bruscoli, Vladimir Calderón, Brett Terpstra, Géraud de Laval, Stephan Burlot, Claudia, and countless others that I hope will not be angry at me for not adding them to this list right now. Thanks, everyone.

The best is yet to come, though; just because, well, paraphrasing Tony Stark in the final scene of "Iron Man", I am the akosma.

Still Learning One Language per Year

Adrian Kosmaczewski

2013-10-06

Quick update about my “one language per year” lifelong initiative:

1992: QBasic
1993: Turbo Pascal
1994: C
1995: Delphi
1996: Java
1997: JavaScript
1998: VBScript
1999: Transact-SQL
2000: C# and Prolog
2001: C++
2002: PHP
2003: Objective-C
2004: Visual Basic.NET
2005: Ruby
2006: LINQ
2007: Erlang
2008: Python
2009: Go
2010: Lisp
2011: Haskell
2012: Lua
2013: C++11

This year it’s time to re-learn C++. The language has substantially changed, and I’m glad to revisit it.

Using Phonegap as a Mobile App Platform

Adrian Kosmaczewski

2013-11-04

This report will provide an overview of the challenges and opportunities brought by PhoneGap when building cross-platform mobile applications for touchscreen smartphones and tablets.

(Download ZIP file[1] with PDF and EPUB version of this report.)

It will be based in the experience of the Trifork team while building real-life applications using this tool.

Introduction

PhoneGap[2] is a mobile development framework produced by Nitobi, purchased by Adobe Systems. It enables software programmers to build applications for mobile devices using JavaScript, HTML5, and CSS3, instead of device-specific languages such as Objective-C or Java. The resulting applications are hybrid, meaning that they are neither truly native (because all layout rendering is done via web views instead of the platform's native UI framework) nor purely web-based (because they are not just web apps, but are packaged as apps for distribution and have access to native device APIs).

The software underlying PhoneGap is Apache Cordova. The software was previously called just "PhoneGap", then "Apache Callback". Apache Cordova is open source software.

History

First developed at an iPhoneDevCamp event in San Francisco, PhoneGap went on to win the People's Choice Award at O'Reilly Media's 2009 Web 2.0 Conference and the framework has been used to develop many apps. Apple Inc. has confirmed that the framework has its approval, even with the new 4.0 developer license agreement changes. The PhoneGap framework is used by several mobile application platforms such as ViziApps, Worklight, Convertigo, and appMobi as the backbone of their mobile client development engine. Adobe officially

[1] report.zip

[2] At the time of this writing, the current stable version of PhoneGap is version 3.1.

announced the acquisition of Nitobi Software (the original developer) on October 4, 2011. Coincident with that, the PhoneGap code was contributed to the Apache Software Foundation to start a new project called Apache Cordova. The project original name, Apache Callback, was viewed as too generic. Then it also appears in Adobe Systems as Adobe PhoneGap and also as Adobe Phonegap Build.

Early versions of PhoneGap required a person making iOS apps to have an Apple computer, and a person making Windows Mobile apps to have a computer running Windows. After September 2012, Adobe's PhoneGap Build service allows programmers to upload HTML, CSS and JavaScript source code to a "cloud compiler" that generates apps for every supported platform.

As presented by Adobe, PhoneGap provides two complementary services:

- A wrapper; PhoneGap packages HTML, CSS and JavaScript files to be deployed and distributed through mobile app stores.
- A bridge; PhoneGap also provides mechanisms to augment HTML5 web applications, allowing them to access and consume information and services otherwise only available to native applications, such as the local address book, the notification system, sounds, and other utilities.

Supported Platforms

At the time of this writing, PhoneGap supports the following mobile platforms:

- Apple iOS
- Android
- BlackBerry
- webOS
- Microsoft Windows Phone
- Symbian
- Bada
- Tizen

Supported Features

As a bridge, PhoneGap allows developers to access the following features:

- Accelerometer
- Address Book
- Camera
- Compass
- File system
- Geolocation
- Media
- Network

- Notifications
- Storage

Not all features are supported in all the mobile platforms in which PhoneGap runs; the chart reproduced in the table below shows the features available in each platform. As a rule of thumb, iOS, Android, Windows Phone, Tizen and BlackBerry 10 support the complete feature set offered by PhoneGap.

Supported PhoneGap Features:

Feature	iOS	Android	WP	BB	Bada	Symbian	webOS	Tizen
Accelerometer	x	x	x	x	x	x	x	x
Camera	x	x	x	x	x	x	x	x
Compass	x	x	x	x	x		x	x
Contacts	x	x	x	x	x	x		x
File	x	x	x	x				x
Geolocation	x	x	x	x	x	x	x	x
Media	x	x	x	x				x
Network	x	x	x	x	x	x	x	x
Notifications	x	x	x	x	x	x	x	x
Storage	x	x	x	x		x	x	x

Design Rationale

The core of PhoneGap applications use HTML5 and CSS3 for their rendering, and JavaScript for their logic. Although HTML5 now provides access to underlying hardware such as the accelerometer, camera and GPS, browser support for HTML5-based device access is not consistent across mobile browsers, particularly older versions of Android. To overcome these limitations, the PhoneGap framework embeds HTML5 code inside a native WebView on the device, using a foreign function interface to access the native resources of the device.

PhoneGap is also able to be extended with native plug-ins that allow for developers to add functionality that can be called from JavaScript, allowing for direct communication between the native layer, and the HTML5 page. PhoneGap includes basic plugins that allow access to the device's accelerometer, camera, microphone, compass, file system, and more.

However, the use of web-based technologies leads many PhoneGap applications to run slower than native applications with similar functionality. Adobe Systems warns that applications built using PhoneGap may be rejected by Apple for being too slow or not feeling "native" enough (having appearance and functionality consistent with what users have come to expect on the platform).

Basic Usage

To create a native application using PhoneGap usually involves the following steps:

1. Create the web application using HTML5, JavaScript and CSS.
2. Install the PhoneGap framework in the development machine.
3. Create a PhoneGap application using the command line tools.
4. Add the required HTML, CSS and JavaScript files.
5. Build the application using the command line tools.
6. Deploy to a device for testing, or to a mobile app store for distribution.

Experience

As advanced as it may be, any tool has limitations, which the developer must be aware of, and strenghts to exploit and take advantage of. This chapter will provide a short summary of highlights and drawbacks encountered while using PhoneGap in real-life projects.

Installation

The first contact with PhoneGap is through the installation of the developer tools. As a toolkit, PhoneGap is entirely dependent on the preexistence of the native developer tools for each target platform. For example, to create iOS applications, the developer machine should be bundled with Xcode, and so on[3].

PhoneGap is bundled through the `npm` package manager system, originally built with the open source project `Node.js`. Through `npm` developers can install PhoneGap in the developer machines. `npm` also takes care of providing updates to the core PhoneGap libraries as they are released.

Command Line Tools In an effort to streamline and simplify the creation and management of PhoneGap systems, Adobe provides a series of command-line tools that can be used to create and build PhoneGap applications.

For example, to create a new application, a developer would write the following command:

```
$ phonegap create --name "AppName" --id com.trifork.AppName foldername
```

The command above would create an application named `AppName` in a subfolder of the current working directory called `foldername`.

[3] This report will not deal with PhoneGap Build, the cloud-based build system provided by Adobe, as for confidentiality reasons some customers might not be comfortable knowing that their applications are being built elsewhere.

Once the application is created, developers can perform several different operations on it; typically, a certain number of plugins would be required, which can be added with the following commands:

```
$ phonegap local plugin add org.apache.cordova.dialogs
$ phonegap local plugin add org.apache.cordova.console
```

These commands not only include the required code in the project (both native and JavaScript), they also modify the permissions and other settings in the platform-specific projects for each target[4].

Similarly, the command line utilities can be used to package and build the native application for a specific platform[5]:

```
$ phonegap build ios
[phonegap] detecting iOS SDK environment...
[phonegap] using the local environment
[phonegap] adding the iOS platform...
[phonegap] compiling iOS...
[phonegap] successfully compiled iOS app

$ phonegap build android
[phonegap] detecting Android SDK environment...
[phonegap] using the local environment
[phonegap] adding the Android platform...
[phonegap] compiling Android...
[phonegap] successfully compiled Android app
```

Dependencies

As stated above, building a native PhoneGap application requires the following tools installed in the local machine (this text assumes that the reader is using a Mac computer running the latest version of OS X and wishes to create iOS and Android applications using PhoneGap):

- Xcode.
- Xcode command line tools (installed separately from Xcode.)
- Android SDK (can be installed through Homebrew.)
- Apache Ant

Challenges

This section will highlight major issues and challenges faced by developers when creating PhoneGap applications, organized in three major groups:

- JavaScript.

[4] It is strongly recommended to install at least the console plugin, which provides a direct connection from the web `console.log()` statements to the native logging mechanism, like the Android logcat or the iOS console.

[5] Unfortunately, at the time of this writing, the PhoneGap Android developer build system does not work when launched from directories that contain a space. Developers should make sure that they create their applications using full paths without spaces.

- Development time.
- Documentation.
- Debugging.
- Integration.

JavaScript PhoneGap applications are built using JavaScript, a language recognized as particularly difficult to master, particularly given the fact that it bears a misleading name. JavaScript is a functional, class-less language with objects, with a single-threaded, run-loop execution model. This description places immediate constraints in the layout and organization of the source code, and many developers, even after having been exposed to the language for years, are not familiar with them.

It is recommended that developers be exposed to a certain level of training (usually one or two days is enough) in JavaScript before delving into PhoneGap applications; this will increase their productivity and will help them debug the resulting applications faster.

This becomes extremely important when dealing with supplementary UI frameworks such as jQuery Mobile or Sencha Touch, which can be run inside PhoneGap applications and provide a higher-level abstraction for creating complex web applications in a shorter amount of time.

Development Time The biggest advantage of having a single code base for all the platforms covered by a mobile solution is, without any doubt, the reduced time required to write and maintain the application. In general, not only is the development time reduced, but also the time spent to fix a bug is also amortized accross platforms.

This principle implies that the true economy of scale is only achieved when several platforms are required for the same source code base. The development time used to write a mobile app in JavaScript is around 60% to 70% the time required to write the same solution in Objective-C, C# or Java. Once this is done, adding a new platform to the PhoneGap solution is a matter of hours.

However, PhoneGap (or any other cross-platform technology, for that matter) does not reduce the time required for marketing, submission to the App Store, customer relationship management, and other related tasks; for each platform, a substantial overhead is expected in non-development tasks, and this overhead is comparable to that of native solutions.

All in all, PhoneGap represents an interesting opportunity as soon as there are more than one platforms required for a mobile solution; otherwise it is impossible to maximize the advantage provided by this approach.

Documentation The documentation provided by Adobe for PhoneGap is probably the weakest point of the whole platform, particularly regarding the exact syntax of the command line options. These have changed through the past releases (particularly between versions 2 and 3.0, and again in 3.1) which makes it quite difficult to move forward in stable ground. The author hereby wishes that the whole toolkit will remain stable for the time being.

In many cases, the syntax of the phonegap commands is simply wrong, and it even changes from page to page in the same documentation set, with errors sprinkled all over. Needless to say, this greatly diminishes the appeal of the whole platform, and makes the development process needlessly complex.

The other part of the documentation, which basically describes the various APIs available to connect to the different device subsystems, however, has been stable for at least 2 major releases, and for the past 3 years; this section of the documentation is solid and clear, and includes lots of useful examples.

The experience shows that these APIs work as intended, even if sometimes it is not very clear that plugins are required to make them work properly. Without those plugins, the application fails silently without any feedback to the developer.

Debugging Debugging PhoneGap applications is another delicate point. While most of the development of a web application happens on a web browser, where the tools are these days acceptably good, solving issues in PhoneGap applications remains a complex task.

Debugging a mobile web application involves the same tasks as in any other platform:

- Setting breakpoints and exploring the behavior of the application at runtime, including variables and call stacks.
- Inspecting the layout of the application, in this case composed by the HTML and rendered using CSS statements.
- Watching the log statements produced by the application immediately.
- Being able to observe the state of the file system or other storage media.
- Performing all the tasks enumerated above in a device, in real time.

Most web browsers these days include excellent developer tools, like Chrome, Safari, Firefox and even the latest releases of Internet Explorer. The problem remains when using native APIs, like the camera or the accelerometer, where browsers do not emulate those capabilities, at least not at the moment of this writing.

In particular, since the release of OS X Mavericks and iOS 7, Safari 7 for Mavericks is apparently unable to properly debug web applications wrapped in PhoneGap containers and running on devices connected

via USB. Several other developers have raised similar concerns, and apparently there is no solution for this problem at the moment.

Another solution for in-device debugging involves using Adobe Edge, a system proposed by Adobe which specifically targets this problem; using Edge, developers can inspect the runtime of their applications directly on the device, getting access to log statements, but without the ability of setting breakpoints in the code, which is a major drawback at the moment.

Integration The final complex point of PhoneGap has to do with the integration with build systems or continuous integration platforms.

In this case, the fact that PhoneGap is bundled as a set of command-line tools helps developers greatly, since the build and deployment of PhoneGap applications can be directly done from build scripts or similar tools (Ant, GNU Make, shell scripts, etc.)

The most important thing to take into account when setting up the build environment is the dependencies; for example, PhoneGap requires Ant and the Android SDK to be able to build Android applications, and so on.

Conclusion

PhoneGap provides an interesting approach to the creation of mobile applications; however there are inherent problems due to the youth of the system and the changing priorities of the corporation behind it.

In the opinion of the author of this lines, it is strongly recommended to avoid PhoneGap for performance-sensitive or complex tasks, like games or productivity applications; the strength of PhoneGap is in the fast delivery of cross-platform business solutions, particularly in the case of form-based applications, used for reporting and data mining.

PhoneGap also requires developers to be comfortable with the JavaScript language and its associated execution model (functional, run loop based, single-threaded, etc.) This skill is usually not found, even in developers having been exposed to the language during years.

Finally, PhoneGap provides substantial benefits when an application targets 3 or more platforms; true economies of scale can be achieved from that point forward, both in development and maintenance times.

Talking to Strangers

Adrian Kosmaczewski

2014-01-10

I'm not the kind of guy that enjoys talking to strangers. It's something quite common and appreciated in Latin cultures, for example to smalltalk about the weather or other inane subject with the person next to you on the bus, or a neighbor, or anything. When I travel alone I prefer single seats, nobody around me, traveling in peace, music, book, writing, thinking. When I walk I prefer silence or music in my ears. When I work I prefer silence. And the respect of this sphere is another of the things that I appreciate in Germanic and Scandinavian cultures, as a matter of fact.

Let's be clear about the fact that I have nothing against human interactions in general, and those who know me agree that I tend to be a rather empathic and friendly person; heck, I'm even a conference speaker and I regularly teach. But I personally don't look for spontaneous meetings, so even if I do not shy away from someone talking to me, I won't usually be the starter of an interaction out of the cold.

And so it happened one day, on my way home from work, I was waiting the train while listening to a Piazzolla track, if I'm not mistaken I think it was "Jeanne and Paul," one of my preferred songs. And what happens with me and Piazzolla is that, whenever I'm in a deserted place with nobody around me (this was an empty train platform in the evening,) I start whistling the song I'm hearing through my earbuds. I can't help it. Particularly with Piazzolla; I like the challenge of the dissonant phrases, the seemingly irrational chords, the sequences that never end, mixing both pain and pleasure to the mix. I just love to whistle Piazzolla tunes. It's relaxing, it's complex and it makes me travel far away.

As my train was entering the station, I cancelled my whistling session as two or three people approached me on the platform, among them an old lady who, just before getting onto the train, spontaneously congratulated me for my whistling. I truly think that I do not whistle properly, and that's why I do this all alone. I know I am often out of tune and time, but I like whistling anyway. I just don't want to impose this to anyone else. So that's why I was surprised, and even more so, because in general I would have expected this to happen in the French or Italian parts of Switzerland, not in the Germanic section.

So I thanked the lady, surprised and smiling, and next thing I knew

we were talking about Switzerland, the weather, the trains, her family and many other of those inane subjects I mentioned earlier.

We arrived in Zürich Hauptbahnhof 40 minutes later, talking and enjoying the moment.

And that’s it.

If you were expecting an analysis, a morale or some kind of afterthought surrounding this story, well, you’re out of luck as there’s none. Although I liked talking to this lady, I still do not start these conversations myself, I still enjoy being left alone, and I still whistle when listening to Piazzolla all alone.

On the Kindness of Nordic People

Adrian Kosmaczewski

2014-02-24

There's this common but, in my opinion, largely unfounded idea among Mediterranean people that Nordics are "cold," so to speak.

I have heard similar arguments in conversations with people from diverse origins such as Italian, Spanish, Greek, Arabic, Latin Americans, French (especially from France), and many others, arguing that Germanic and Scandinavians are cold hearted, blissfully devoid of emotion, ready to throw you and your sensitivity overboard, just like a Viking would throw the crufty corpse of a prisoner to spend holidays in Valhalla. That, in a sharp contrast with their own vision of themselves, that is, "us shiny happy people of the Southern Sun©®™ where we have nice tomatoes and samba and soccer players and surgically enhanced fucktoys on TV and freshly harvested economic crisis every season and shantytowns the size of Coruscant and temperatures comparable to those of a Tattooine summer."

Let me debunk this overly simplistic and myopic perspective thanks to my own, simplistic and overly myopic, largely non-scientific and extremely debatable observations. What follows is a profusely cynic, politically incorrect, adverbially ridden text, laden with lots of clichés and other nuisances, so all of you rigid mindsets or uptight characters, you might want to stop reading now.

My theorem is that the common Latin wisdom misleadingly exchanges the concepts of empathy and hypocrisy. In particular, Latin people are unable to distinguish between both, and as a corollary of this rule, they fall into the common trap of trusting the wrong kind of people once and again.

To demonstrate my theorem, first, three axioms:

1. There's shitty people everywhere, just like there is nice people everywhere.
2. The ratio of shitty vs. nice people in any given society is a constant, throughout space and time in every country, no matter the latitude, average income, median shoe size, total number of World Cup wins, flag colors (CMYK and/or RGB), or Moody's credit risk rank; I hereby name that constant the Number of Kosmaczewski (I always wanted to jump into posterity with some bold contribution like the last one.)

3. There are only two axioms.

The next step is a Gedankenexperiment to pay some tribute to Einstein and establish a sense of relativity and existential complexity. In this experiment we are going to clone ourselves in three entities, and each of which will migrate to a different section of the world. The first clone will move its ass to Argentina, arguably a largely Mediterranean country; the second lucky chap will land in the German section of Switzerland, arguably a place full of cold-hearted SOABs; and the third will serve as a "witness," and will be sent to the Soyuz Station in Anctartica, an abandoned Russian scientific station located at 70Â°34'52"S and 68Â°47'08"E, in which the sheer solitude, the outstanding amount of empty bottles of vodka and the bitter cold will preserve this clone in a relatively stable state, if not in a dangerous level of hypothermia, until global warming does its job of bringing it back to life in the next few years.

The conclusion of this flawed, short, uncanny, unprovable and improbable experiment is that all clones empirically verify the value of the Number of Kosmaczewski and henceforth conclude that we are all wrong, except me, the author of this brilliant piece, 'f course.

You're welcome.

NB: There are no comments enabled in this blog, and with reason. If you don't like what I say, well you can always spread your venom over Reddit. Be my guest. You'll excuse me if I don't pay attention to your rambling, I have more writing to do.

Best Books of 2013

Adrian Kosmaczewski

2014-04-06

It is that time of the year again, just like in 2012[1], 2011[2], 2010[3], 2009[4], 2008[5] and 2007[6].

Hoping that you are going to notice that the “non technical” section is taking precedence over the “technical” section, these are the books I’ve read and written in 2013 (in no particular order):

Non-technical

- Estados del Alma[7], the first poetry book published by my sister and lifetime friend Laura Fähndrich.
- Man and his Symbols[8] by Carl G. Jung.
- Norwegian Wood[9] by Haruki Murakami.
- Clases de Literatura[10] by Julio Cortázar.
- Octaedro[11] by Julio Cortázar.
- Masters of Doom[12] by David Kushner.

Technical

- iOS Storyboards[13] by Daniel Steinberg.
- Presentations in Action[14] by Jerry Weissman.

[1] /blog/best-books-of-2012/
[2] /blog/best-books-of-2011/
[3] /blog/best-books-of-2010/
[4] /blog/best-books-of-2009/
[5] /blog/best-books-of-2008/
[6] /blog/best-books-of-2007/
[7] http://edicionescardenoso.blogspot.ch/2014/01/estados-del-alma-de-laura-fahndrich.html
[8] http://en.wikipedia.org/wiki/Man_and_His_Symbols
[9] http://en.wikipedia.org/wiki/Norwegian_Wood_(novel)
[10] http://www.amazon.es/CLASE-DE-LITERATURA-Biblioteca-Cortazar/dp/8420415162
[11] http://www.amazon.com/Octaedro-Spanish-Julio-Cortazar/dp/9505111835
[12] http://www.amazon.com/Masters-Doom-Created-Transformed-Culture/dp/0812972155
[13] http://editorscut.com/Books/004iosstoryboards/004ios-storyboards-details.html
[14] http://www.amazon.com/Presentations-Action-Memorable-Presentation-Lessons/dp/0132489627

- Async JavaScript[15] by Trevor Burnham.
- Essential iOS Build and Release[16] by Ron Roche.
- The Past, Present, and Future of JavaScript[17] by Axel Rauschmayer.
- The C++ Programming Language, 4th edition[18] by Bjarne Stroustrup.
- A Tour of C++[19] by Bjarne Stroustrup.
- Android Cookbook[20] by Ian F. Darwin.

Mine

- Sencha Touch 2 Up and Running[21] by yours truly.

[15] http://pragprog.com/book/tbajs/async-javascript
[16] http://shop.oreilly.com/product/0636920022282.do
[17] http://www.oreilly.com/programming/free/past-present-future-javascript.csp
[18] http://www.stroustrup.com/4th.html
[19] http://www.stroustrup.com/Tour.html
[20] http://shop.oreilly.com/product/0636920010241.do
[21] http://shop.oreilly.com/product/0636920026877.do

To Never Forget

Adrian Kosmaczewski

2014-05-06

In 1992, a section of the football stadium at Bastia, in Corsica, fell right before the beginning of a match. Tens of people died and hundreds were injured when a badly constructed section of the stadium fell under the weight of the supporters cheering for their team. The families of the victims have asked that no football matches should be played in France ever again on every May 5th thereafter, in commemoration of that date and that event, for respect of the victims and to never forget.

In September 2001, a terrorist attack caused the loss of almost three thousand people in New York City. Since that day, security has heightened, all events in our lives are being closely monitored by government agencies, air travel has become a burden and a chore. International coalitions torture and deport innocents to Guantanamo in the name of peace and cooperation. The families of the victims have asked for revenge and for security, in commemoration of that date and that event, for respect of the victims and to never forget.

In 2004, a well-known disco in Buenos Aires called Cromagnón, ignited in fire killing many kids who were listening to their favorite band, Callejeros. The parents of those kids, asking for justice (or revenge, or both) stopped all traffic in 2 blocks of the street where the disco was located, and this during years. The whole life of that neighborhood was changed forever; shops had to close, bus lines had to be changed, TV cameras were located there 24/7 ever since. The families of the victims have asked that no vehicle should circulate on that part of the city ever again, in commemoration of that date and that event, for respect of the victims and to never forget.

See the pattern?

At this rate, soon the whole of mankind will stop moving, breathing and doing anything. Because every day, somewhere, a tragedy (or a commemoration thereof) takes place. The history of mankind is full of events that are disgusting, filled with gory and terrible details, showing that we are far from being a caring species. We just do not give a shit for anything or anyone. Of course, many religions pretend the opposite, and make their followers believe that they are somehow different, that they will be saved.

In the meantime, for the sake of memory, we cannot build new buildings because all buildings are historical landmarks. We cannot speak our minds in respect of some healing memory. We must censor ourselves 20 times before saying anything.

And then we wonder why kids kill themselves. Why people do drugs. Why alcoholism is still killing so many people. Why our lives are so miserable.

I could fill an entire calendar with sad events from my own life, and then I could lead a battle asking everyone to respect the mourning of my countless, painful losses. Each one of us could do that, and then we would argue as to which of our pains is the greatest.

World championships of pain would be played, people would talk about their miseries on big screens and others would vote on Facebook or Twitter for the worst and most memorable of those shitty memories. The champion would win the right to make everyone else in the world to stop breathing for a moment, in commemoration of that date and that event, for respect of the victims and to never forget.

We have to learn to let go. Let go of the pain, cry it for yourself one last time, pat yourself on the shoulder because nobody is going to do it, and learn that each and every one of us is suffering the chore of living a life.

We have to learn to let go. We do not have to forget; but we can and must move on.

Global Conscience

Adrian Kosmaczewski

2014-06-06

Our objective is global awareness. It is our ultimate goal.

Today we have far more ways to know what is going on in the minds on the other side of the planet, than in those of the people sitting in front of us in the train.

But we will get there, too. Each one of us will be totally, instantly, fully aware of each other. It will happen. And I mean it in a big way, in a holistic way, including every detail. Including even the details that we will not want to share at first.

Some scientist will learn how to interpret those electric signals in our brains, so that they can be translated into images and words, palatable to anyone else, in any medium they choose. Or even easier, the signals from all brains will be broadcasted, like a massive Twitter of thoughts, to whomever wants to listen. No need to translate; just plug and play.

We will know everything about everyone. At any given time. In any language. And all of that will be recorded, and it will be used against us, of course. It will generate shockwaves of anger and happiness. People will discover each other. Psychologists will finally be understood by the rest of us. Frauds and heroes will surface naturally among a sea of human stories. No omission will pass by undetected, no dream will be left without analysis.

No translator will be needed ever again. The pure language of thought will flow among us, and thought deprivation will be considered a felony.

Maybe even censorship will become not only unlawful, but actually impossible.

Learning processes will approach the usability seen in a Matrix movie.

No love story will remain hidden.

All of this will happen automatically, instantaneously, by everyone, and everywhere, at any time. Fear and pain of anyone will be felt by all of us, simultaneously, at once.

Empathy will no longer be an option, but a core feature of our species.

This is something we are actively looking for. This is what we want. We want to know, and we will know. The question is, are we mature enough to handle what we are going to see when this happens?

Graffiti

Adrian Kosmaczewski

2014-07-20

Quiero morir, quedar en el olvido.
Sería como abrir la tapa de un libro,
leer la primera hoja
y quedarse dormido.

(Graffiti seen in a wall in the Vergara street, in the neighborhood of Vicente López, Argentina, around 1985.)

The Tao of Swift

Adrian Kosmaczewski

2014-09-09

I gave this talk in September 2014, in Zürich, the same day Apple announced[1] the Apple Watch.

Tonight is a night that is really interesting for Swiss developers. First we are going to learn about Swift, whose name is similar to that of the "Society for Worldwide Interbank Financial Telecommunications" which is a very well known institution in Switzerland. Anyone working in a bank? And then we're going to laugh at Apple trying to sell a watch that crashes and which runs with batteries that last only for 6 hours. So, what we need is a bit of cheese and to dress in military costumes, and this would be a truly 100% Swiss evening.

Who has been using Swift in the audience? Well, this is an introductory talk, so there might be some things that you already know. Feel free to correct me if I say something wrong during my presentation, as we have all been exposed to the same information in the same amount of time. I hope I can, however, bring something new to your knowledge.

Introduction

Let me quote the Tao of Programming, to bring back some context:

> "Thus spake the Master Programmer: The Tao of Programming flows far away and returns on the wind of morning. The Tao gave birth to machine language. Machine language gave birth to the assembler. The assembler gave birth to the compiler. Now there are ten thousand languages. Each language has its purpose, however humble. Each language expresses the Yin and Yang of software. Each language has its place within the Tao. But do not program in COBOL if you can avoid it."

If the world followed the Tao, Apple would make the hardware, Microsoft would make the developer tools, and everything would run under QNX. Alas, QNX has been bought by BlackBerry, Microsoft makes operating systems and Apple makes developer tools.

[1] https://www.theverge.com/2014/9/9/6125873/apple-watch-smartwatch-announced

Welcome to our world. A world where the Tao is not being followed closely, where we have lost touch with the essential stuff.

Let me start this presentation by introducing myself. I am Adrian Kosmaczewski, and my two primary functions in this world is to write and to talk. I write code and books, and I talk a lot. I also come from a place far, far away.

Now let me introduce you to Xcode 6. This is the tool, currently in its seventh beta iteration, that allows you to crash very often. Whenever you need to crash, Xcode will be there to help you do that quickly and efficiently. The funniest thing about Xcode is that after it crashes it displays a dialog box that says "Xcode quit unexpectedly" which as everybody knows, it's just not true at all. We all know Xcode is going to crash. The question is "when."

Xcode 6 includes a gazillion new features, but there's one I will use tonight extensively: Playgrounds. These are like REPLs (Read-Eval-Play-Loop) but with all the crashing power of Xcode and none of the awesome autocompletion, refactoring, embedded help and other usability and productivity features of Xcode, which are actually very few.

Playgrounds give you very weird error messages, and then as soon as you try to help it, it crashes. So, there you go. Of course, this makes people very angry.

Warning

The same applies now to Swift. As much as Swift is a beautiful language, and we are going to see that closely tonight, I have to be blunt and clear with you right away: Swift is not ready for prime time. The current version of Swift still lacks many important features, for example there are no class variables as of now; and even worse, there is a project in GitHub that clearly states many known valid language constructions (at least valid following the specifications already available) which make the compiler crash.

So, before I begin this presentation, a warning: do not use Swift in production applications right now. But, should you spend time learning it? Yes, indeed, I think it is a good idea. I actually try to learn a new programming language every year. But keep in mind that everything you are going to learn might change, and that if you write production code with it right now, you will have to rewrite it to comply with future changes in the language.

And if you do not believe me, here's Erica Sadun and Rob Napier saying the same. So, you've been warned.

The beauty

However unstable and incomplete, Swift is a beautiful language to write applications in. It is as simple and approachable as Ruby and

Python, it has complex features borrowed from more complex languages like Haskell and C#, and it compiles into native code, which is executed without intermediaries, and some compiler errors and warnings are very similar to those from C . So, yes, the language borrows features from everywhere.

Apple says that Swift is modern, safe, fast and interoperable.

“Swift is modern, because it has closures; tuples and multiple return values; generics; fast and concise iteration over a range or collection; structs that support methods, extensions, protocols; and functional programming patterns, e.g.: map and filter.

Swift is safe, because variables are always initialised before use, arrays and integers are checked for overflow, and memory is managed automatically.

Swift is fast, because using the incredibly high-performance LLVM compiler, Swift code is transformed into optimised native code, tuned to get the most out of modern Mac, iPhone, and iPad hardware.

Swift is interoperable, and can be used together with Objective-C in the same project.”

Sounds too good to be true, huh? Yeah.

Finally, Swift has a beautiful name that make Google requests extremely hard. This is because Swift is not only this language, but also:

- A bird
- A fox
- A singer
- A car
- A protocol for interbank communication
- And another, completely unrelated, programming language

Although, to be fair, in terms of naming programming languages, we’ve seen worse.

Reason

Why Swift? Well there are many reasons; to begin with we can all agree that Objective-C as a language felt outdated. It’s basically a good old sitcom from the 80’s that reruns every day on the screen of our laptops. Meanwhile, lots of different languages have captured the imagination of developers everywhere, so we can say that we needed to change.

Another reason is that, well; many developers simply disliked the whole idea of a language with pointers and weird square brackets all over the place. But that’s an opinion, and of course the Interwebz is filled with them.

Who made it

Swift was created by Chris Lattner and his team at Apple, who worked on the language as a secret project since 2010. The announcement of the language came as a surprise during the latest WWDC keynote in June, so everything that we know about the language has been discussed in the past 3 months.

To bring a bit of perspective, Chris Lattner is the chief developer of the LLVM project, which brought as many nice things, such as the CLang Static Analyzer and also, of course, Automatic Reference Counting (ARC.)

The language jumped to the 16th place in the TIOBE ranking in July, then dropped to the 22nd in August and right now it is in the 18th place. Given the amount of interest given to the language online, it is clearly becoming a very interesting topic on the internet.

Interoperability

Swift has been thought from the ground up as a transitional language from Objective-C. Swift classes can inherit from Cocoa classes, and Swift can use any Cocoa API. Inversely, Xcode makes available Swift classes to Objective-C, and both languages can (more or less) coexist on the same project.

However, the following list of Swift features are not available in Objective-C: in no particular order, generics, tuples, enumerations, structures, top-level functions, global variables, typealiases, variadics, nested types and curried functions do not work in Objective-C. Which clearly leaves out lots of nice Swift features.

How does Swift compare?

Apple markets Swift as “Objective-C without the C,” but given the previous slide, I guess you can imagine that Swift is really different from Objective-C. So, what does it compare to?

Programmers love to talk about how Java was inspired by C , or about how NewtonScript inspired JavaScript or not. The truth is, this does not matter, but for the sake or throwing flames, here is what we know about Swift:

- Its syntax looks incredibly similar to Scala.
- It has lots of features borrowed from C#.
- It does not suck too much like JavaScript.
- And apparently it feels a lot like Haskell.

But the Haskell comparison is the one that won; even Chris Lattner stated in his home page that Swift took lots of inspiration from Haskell, and this has led the whole internet community to rejoice.

I have made my own investigation, and indeed, there was a certain Charles Haskell Swift, born in Massachusetts in 1838; then I found out

that you can buy a Suzuki Swift in Haskell, Oklahoma; not only that, but both Taylor Haskell and Taylor Swift have Facebook profiles; I can safely say that there is a clear connection in there.

So, Haskell developers; apparently you are going to love Swift. And Apple is wrong; Swift is not Objective-C without the C; it is more Objective-Haskell, actually.

Features

So, what features does Swift bring to the table? Well, all those you have seen in the previous slide plus the following:

- Strong typing
- Type inference
- Closures
- Optionals / Nullable types
- Generics
- Custom operators
- Tuples
- Interoperable with Cocoa
- Changes every week

Is Swift functional? Not really. Although the language could have been designed as a purely functional language, it is not; to state it simply, it allows for side-effects and mutable values, and it also lacks many functional libraries usually available to operate on lists recursively. Even more, Swift has to operate and work with the Object-Oriented world of Cocoa, so clearly Swift is a transitional language, which will take us, app developers, from an Object-Oriented Cocoa library to a new one, probably based on functional concepts.

So, here is my blunt statement: “Cocoa is the new Carbon.” The standard library brought by Swift will, I think, grow in the future, and probably allow for 100% pure Swift implementations, without requiring the use of the Cocoa Frameworks.

But, this is an opinion, of course. You never know with Apple.

In any case, Swift is built for speed. One of the biggest goals of Swift is to provide a high-level language that provides raw metal performance at the lowest possible level. It has abstractions of sufficiently higher level for getting things done fast, but it also brings malloc() and pointer manipulation in case you need it. Right now, many early tests show that Swift code is already faster than the corresponding Objective-C code.

Instantiation

Let me open a Playground. An Xcode Playground allows us to write code and to see it crash in real time. It is a practical implementation

of the “fail fast” mantra; if you are going to fail, you’d better fail as fast as possible.

So, let us start by importing some code. I am going to import my own framework, one that I’ve created for this presentation (yes, the code will be available in GitHub, of course) so I can show you some cool features of the language.

So I create a Playground by selecting “File - New - Playground” in Xcode, all while I pray for Xcode not to crash on me. I save the playground on my Desktop, because that’s where I save most of my stuff anyway.

The first thing I’m going to type in my playground is import PresentationKit. Most frameworks in OS X and iOS have the “Kit” moniker, so when I created mine I figured out that I might as well use that suffix, too.

The first thing that strikes about Swift is the lack of semicolons. In Swift semicolons are really optional, unlike JavaScript that tells you the same until you minify your code and your application breaks. In Swift you only need semicolons if you have two statements in the same line; otherwise, hold the semicolons.

After I import my framework I can start to use my own classes in it. I’m going to create an instance of my presentation slides, and for holding those slides in memory I am going to create a variable. How do you create a variable in Swift? Just like in JavaScript: var presentation.

The keyword var does not convey, as you might imagine, any type information with it; every variable is declared using the same type.

The next thing I’m going to do is to create a new instance of a class I have defined in my framework. The syntax is the same as in Python, and I just have to write: var presentation = Presentation().

So now I have a presentation variable holding a reference to an instance of the Presentation class in memory. The Swift code above is technically the same as the following Objective-C code:

```
Presentation *presentation = [[Presentation alloc] init];
```

In Spanish there is a saying that goes “Lo bueno, si breve, dos veces bueno,” which is roughly translated as “What is good, if brief, is twice as good.” Well, not roughly because I was born in Argentina and Spanish is my first language, so I can assert that the translation is correct.

Type inference

One of the most fundamental features of Swift is type inference. We do not need to explicitly provide type information to the presentation variable, as we leave the Swift compiler to “guess” that for us. Unlike in JavaScript, variables in Swift are strongly typed; you cannot assign

a String to a variable that has been used to hold an Array previously; Swift will complain about it!

```
var presentation = Presentation()
presentation = “some string” // –> this yields a compiler error
```

Swift automatically infers the types of the variables at compile time and will use that information to make sure that we do not screw things up, which is one of the basic capabilities of human beings. (minute 9)

Constants

Instead of using the var keyword, we could have used the let keyword, and that would have created a constant value instead of a variable value. Why they chose let instead of const is beyond me.

One of the nice things about let is that it brings a smaller version of a very nice feature of C , namely “const correctness.” Any C developers in the room? You know const correctness, is when you have method signatures that go like const string const * methodName() const and you have a lot of const keywords everywhere, and each has a particular meaning; but the one I talk about is the one at the end, which states to the C compiler that this method is not allowed to have “side effects” on the current instance. And this is very important, as it allows the compiler to optimise things a little bit, and brings Swift closer to the functional programming world.

When using the let keyword, Swift will similarly block the objects assigned with it to be mutated – but of course there is a mutating keyword that you can use in case you want to make your life even more miserable.

Functions, Methods and Closures

Swift can be seen as a “federation of languages,” a bit like Scott Meyers has described C ; you can use Swift as:

- A procedural language
- An object-oriented one
- A functional one

There are some “metaprogramming” features introduced by Generics, but at least in the current implementation they are very limited, particularly compared to what Templates have brought to C .

Functions are a basic block in Swift. Actually, Swift functions are blocks, in the Objective-C sense of the word; you can pass a Swift function whenever you see an Objective-C API that expects a block, but without the “caret” (^) character everywhere. That’s already a bonus.

Even better, functions in Swift are closures AND first-order objects; you can pass them as parameters, return them from other functions,

and even better: class methods are curried functions. So functions are a very important part of the life of a Swift developers, and it is very important to know them well.

To define a function, use the func keyword:

```
func functionName(parameterName: InputType) -> OutputType { }
```

Following the tradition of Objective-C, Swift functions have parameter names, which is one of the best features of our old beloved language. You do not have to guess the semantics of the 4th parameter of that function; just read the name and you'll see what it is about.

Functions can be overloaded; you can have several functions with the same name, as long as they take (or return) different parameter types.

Optional Types

This is also a very important feature of Swift. Any type in Swift can be defined as optional, just by using the ? interrogation sign after it. What does it mean? It means that the type can be nil or not; which is the default behaviour of pointers in C, C and Objective-C. When a type is specified as not optional, it cannot handle a nil value, and the application will crash if that situation arises. This safeguard helps app developers uncover "null reference" situations as early as possible.

Unicode

This is arguably the most important feature of the language. Unicode variables and values allow for incredibly stupid code, which is actually extremely hard to write and read, but which looks great in a small handful of editors that are able to handle Emoji correctly.

So, in all of its glory, let us check some examples of this powerful feature.

Structs, Enums and Classes

Swift has Classes, of course, which follow the classical pattern of Objective-C and Java; single inheritance with protocols. You have only one base class, but you can implement a large number of protocols.

Class instances are instantiated on the heap and are passed as references; they work exactly just like in Objective-C. However Swift has Structs, which are instantiated on the stack and passed around by copy. They are much more powerful than C structs, in spite of the name, and they can comply with protocols (that is, implement interfaces in Java speak) and have methods, just like classes do.

Another difference between classes and structs is that the latter cannot use inheritance; this means that structs are well suited for small…

structures of data that you want to pass around as values: points, vectors, coordinates, etc.

Enums in Swift are a completely different beast; they can have associated values, and developers can convert from and to those associated values. Enums also support methods, and can implement protocols, which make them extremely powerful. They are used to describe entities who have a limited number of possible values.

Protocols, Extensions and Generics

Protocols are to Swift what... @protocol are to Objective-C, or interface to Java and C#; in pure C speak, they are abstract classes with pure virtual methods. They allow for inheritance of interface, instead of inheritance of implementation, like the model provided by C .

Extensions in Swift are equivalent to Objective-C categories, that is, a mechanism used to extend existing types without using inheritance. They are extremely useful as well to separate the implementation of a class into distinct files or chunks, allowing developers to clearly separate concerns and to organise the code clearly.

Finally, generics in Swift are very similar to their counterparts in C# or Java; for the moment at least they do not allow for the creation of the same structures as those allowed by C templates, such as those described by Alexandrescu.

Custom Operators

Ah, custom operators. I can hear C developers screaming. Please, C , scream. Yes, we have custom operators in Swift, and they are weird beasts. They are defined globally, outside of any class, and not inside the class on which they operate. There can be operators infix (that is, in between values), prefix and postfix (which are obvious, I suppose.)

Interoperability

You can mix and match Swift and Objective-C classes in your project, however Swift is much more advanced and many of its features are strictly not compatible with Objective-C.

A Sample Application Using WebKit

I'm going to end this presentation by building a small application on the command line, this time not using a Playground but actually using Vim and iTerm and tmux and the command-line compiler. Because, you know, I'm rebel that way.

Many things I have not talked about

- Standard library
- WillSet / DidSet in setters
- Memory management in closures
- Convenience initializers
- Lazy properties
- Nesting of classes, enums and structs
- Pattern matching in switch statements
- reStructuredText documentation headers
- Default parameter values in functions
- Monads, futures, promises and other functional programming constructions
- Nested comments

Papanicolau

Adrian Kosmaczewski

2014-09-20

Todo arrancó hace un rato. Como todo. Porque todo arranca en un rato. Algún rato. Le digo a Clau. Muchacha, que lloras, que ríes, que pataleas, hacéte un papanicolau. Porque de acá cien años les dirán a las mujeres de hacerse un karolvoitila o un jorgebergolio también, quien sabe. Entonces le dije (a Clau, no al papa) no sabés quien es Sapag? Y no me dice, le digo me dice no sé vistes le digo me dice me dice le digo no sé, entonces aparece el espectro del iutús y sus videos horribles que son básicamente un extracto de VHS numerizado en una PC con un Windows 3.1 trucho que me dieron acá a la vuelta, y el VHS era la copia de uno que me paso mi tía que le había pasado su prima de Olavarría, donde la señal del Canal Nueve Libertad llegaba sólo las noches de luna llena. Bueno, así y todo te lo suben al iutús, y ahí aparece Hugo Guerrero Marthinheitz que ni idea como se escribe el apellido y mira que yo de eso conozco. De apellidos complicados, digo, y Clau se durmió y yo acá luego hago click y pum, un video con Jorge Donn. Y digo, como llegué acá? Ah, si, Tita de Buenos Aires.

Fiuuuuu.

iBeacons

Adrian Kosmaczewski

2014-10-26

Presentation about iBeacons, given on May 26th, 2014 in Zürich (CH), August 13th in Leeds (UK) and August 14th in London (UK), and in October 2014 in Durban (ZA).

Welcome to this presentation about iBeacons. I'm thrilled to have you all here tonight, thanks for coming; the response to this talk has been phenomenal and I couldn't be happier to welcome you to this event.

There will be a question and answer session at the end, so please make sure that your smartphones are set to "not disturb" or "airplane mode" and please wait until the end of the presentation to ask any questions you may have.

Introduction

I'd like to start this presentation by showing you a short movie. Because, I think it's nice to start a talk by telling a story.

Who remembers this film? Of course, it is "Minority Report," a movie by Steven Spielberg from 2002. And a rather prescient movie, that is, for many of the technologies shown in the movie are based on solid visions of a possible future, as foreseen by a cast of technical advisors who helped Spielberg to make the movie. Spielberg actually hired a team of experts to get a clear idea of what the future might look like in… 2054.

The story happens in the year 2054, so let me tell you this, now: "Welcome to the future."

Because the idea behind iBeacons it's exactly what you just saw in this short movie. And so the objective of my talk tonight is to help you bring an answer to questions such as "What is iBeacon?", or "How does iBeacon work?" or even "When should I use it?", or even better yet, "When should I not use it?" And not only that, but also "Where can I buy iBeacons?", "How can I use iBeacons?" and finally the very important one, "Why should I care?" — so, yeah, lots of questions.

To answer all of those questions, and to give you a frame of reference for your understanding, I'm going to describe iBeacons using four dimensions:

1. Marketing
2. Technology
3. Usability and User experience
4. Security and Privacy

Each of these dimensions will provide us with a focal point where we can direct our attention, so that we can understand the implications of this new technology. I guess each of you are responsible for at least one of the bullet points in the slide behind me, so I hope that this overview will give you a much required perspective of the whole iBeacon thing.

Marketing

So let's start first with the "Marketing" aspect. How many of you are in charge of marketing tasks in your companies? What are the main tasks that you have to deal with? Set up campaigns? Engage your users? Get feedback from them?

The official definition of marketing is the "process of communicating the value of a product or service to customers, for the purpose of selling that product or service."

Making the customer buy our products takes us to great lengths. Lots of different strategies exist, all leading to the great holy grail of Marketing, where the product is so specifically tailored to the customers' needs that they just cannot avoid getting it.

I am using here the word "Marketing" in the most generic sense, making a direct reference to all of its related concepts, such as product marketing, pricing, distribution, service, retail, brand management, account-based marketing, analysis, research, segmentation, strategy, social marketing, and of course identity.

In order to increase the interest in your product, to customise and to adapt a product, the first thing you have to do is to know the customer. Or at least to be aware of this magic word called "Context."

And Context has a lot to do with location. Knowing where the user is gives marketing teams a strong foothold, but until now, location information was hardly of any utility when indoors. Thanks to those GPS chips in our pockets, it's become very hard to lose oneself in the wilderness, so to speak. We know where we are at any time. Heck, we even know where our family and friends are at any time, but we could not, until now, pinpoint locations correctly in short distances or in indoor situations.

The basic premise of iBeacons is to provide a simple solution for low-range location services. They are an IPS: an Indoor Positioning System (or Micromapping system,) like many other similar systems, by the way.

iBeacons enable very interesting scenarios, because they can relate

to any kind of marketing activity, such as advertising, branding, direct marketing, personal sales, product placement, publicity, sales promotion, loyalty marketing, mobile marketing, premiums, prizes and much more.

For example. Imagine that you are in a clothing shop; how about getting information about those shirts, or a special notification about a spot offer on socks, or some quick information about the latest promotions, or the quality of a particular clothing item? How about checking out quickly and securely as soon as I arrive to the cash register?

Take a museum for example. What about ditching those clunky museum guided tours by a more on-the-spot experience, where you actually receive information about that author, period, work of art or special exhibition as you move close to it? What about having your museum application automatically giving you the contextual information about the period in Roman history where the sculptures in the current room have been made?

Of course the notion of “indoors” is vague; what about a stadium? Of course we can use the GPS in it, but even with all the precision of our satellites out there, it is still very hard to know if you are close to seat B–134R, or if the closest emergency exit is at your right or your left, or if the closest bathroom is less than 5 minutes away by foot, or where the closest hot-dog seller is.

The same ideas could be applied in conference centres, and at Trifork we know a lot about that, because we have our own GOTO conferences all over Europe; who is the next speaker in this room? What is the name of current speaker in this room? What is the wifi password in this room? Where is the nearest emergency exit? And so on and so forth.

But let’s move further away; what about music festivals, like the one in Nyon near Geneva? I don’t know if you’ve been to the Nyon Paléo Festival, it’s an incredible festival held every year in July with great artists. It happens in a large field outside of the city, where usually 3 or 4 acts happen at the same time in rather muddy conditions (don’t forget your rain boots if the weather is nasty!) So, in that context, many questions arise, and iBeacons can provide interesting answers too. For example, what’s the name of that singer in the main châpiteau of the Festival? Where’s the nearest toilet? (that’s an important question after some glasses of your favourite drink) Where’s the emergency room? (an important question after far too many glasses of that drink of yours) Where’s the nearest kebab shop? (don’t forget to eat while you drink!)

Finally, even more important: what about hospitals and healthcare? How about doctors being able to get instantaneously information about a particular patient just by getting closer to the bed? How about family members being able to locate the closest waiting room in that huge maze of buildings, elevators and hallways that are modern hospitals? How about being able to quickly pinpoint the

location in the building of Dr. Smith, who happens to be the only available expert in heart surgery right now? Can you imagine the sheer number of episodes of ER and Grey's Anatomy such scenarios would yield?

Seriously, iBeacons enable these and many other interesting scenarios, in which consumers, visitors, public and attendees can get contextual information, maximising the value they get out of "they being there" at that moment of time.

On the other side of the equation, of course, we have the magic word "Analytics" coming up. Gathering all the information you have from the movements of your users "in situ" you can know, in real time, the most popular places, the most visited, the most interesting, get instant contextualised feedback from users at any given time, just by tracing their physical movements in the premises of your mall, shop, stadium, museum or conference centre, all of this leading to increased fidelity and retention of your customers.

Indeed, all of this sounds like Nirvana.

I hope you all agree with me in the extraordinary disruptive potential of such a technology, and I hope that you are all, just like I am right now, wondering about the countless possibilities, as well as the important risks that such a knowledge brings.

Because, you know, with great power comes great responsibility.

I am going to talk about some of the risks involved tonight, but for the moment suffice to say that it's quite incredible to think that all of this was very quietly introduced by Apple last year, and that it is already available in most of the world's pockets, where the tools are already installed and waiting for you to come up with ideas.

Technology

Now let's talk about Technology. How are iBeacons implemented? What are iBeacons?

The most important thing to know about iBeacons is that they are based on an open standard: Bluetooth 4.0, also known as "Bluetooth Smart", is a standard since June 2010. Bluetooth 4.0 consists of three protocols, "Classic Bluetooth" for backwards compatibility, "Bluetooth High Speed" based on Wifi standards, and "Bluetooth Low Energy" also known as Bluetooth LE.

"iBeacon" is an open standard based on Bluetooth LE, released by Apple, and in no way it is restricted to Apple devices, such as MacBooks or iPads. Any device that has the required Bluetooth LE hardware and software can interact seamlessly with iBeacons.

Apple has trademarked the name "iBeacon" and has extended the MFi program aimed to external hardware manufacturers, in order to be able to sport the official iBeacon logo shown in the slide behind

me, and to ensure a minimum level of compatibility among manufacturers.

But what about NFC, or “Near Field Communication” as it is known? NFC is another IPS (Indoor Positioning System) that is fairly popular in the Android world; indeed both BLE and NFC can be seen as competing technologies, but there are a few key differences between them:

1. BLE has a much broader range than NFC, up to 50 meters in ideal conditions, usually a maximum of 30 meters;
2. A much faster data transfer speed;
3. Really faster setup time;
4. About the same power requirements;
5. BLE offers cryptography off-the-box, while NFC may not;
6. Both use a different frequency range.
7. In terms of their “nature”, NFC tags are mostly passive, which means that they just reflect the signal emitted from the smartphone, just like passive RFID tags; on the other hand, iBeacons are active, they emit their own signal thanks to an embedded power source. Which leads to…
8. A much higher cost. The price difference between NFC and iBeacons is big, indeed.

From a pure technical standpoint, iBeacons themselves can be thought of just as small emitting antennas using Bluetooth LE signals.

The beacons themselves, and this is very important, do not contain any logic or are in any way “aware” of their surroundings. They only broadcast, 24/7, a fixed signal to their immediate surroundings with a 2.4 GHz frequency, just like any other Bluetooth device would do.

Let me repeat this point: **iBeacons do not push anything to your device; it is your application that triggers the notifications and actions to be displayed at any time.** Just like a lighthouse in the shoreline, an iBeacon broadcasts a simple signal to whoever wants to listen; it is up to the captain of the ship (in this case, your application) to correctly interpret the signal and to take the right decision.

Thus, what contains the logic and intelligence about beacons are either Apps (distributed through the usual channels, like the App Store or similar) and Passbook items (like coupons or event tickets) both of which can “listen” to iBeacons and prompt the user to perform activities or to provide them with additional information. This is by design, to ensure that the end user is always ultimately in control of the user experience, and is able to get rid of annoying applications at any given time.

What iBeacons emit is three pieces of information:

1. A “unique universal identifier” which is a machine-generated, weird sequence of characters that is guaranteed to be unique across time and space;

2. A “major” integer number;
3. A “minor” integer number.

UUIDs look like shown in the slide, and there can be at most 3.4 × 1038; this number is so large that you could have created 7 x 1020 UUIDs every second since the Big Bang and there would be still new values coming up.

How are these values used? Let’s look at an example, for example this typical corner of a typical city like New York with all of its isometric buildings. Let’s say that a small coffee shop store has a couple of branches throughout the city; the first branch, the main branch, would start the rollover of iBeacons and would then choose a UUID for itself. The first iBeacon, thus, would have that UUID, and a major and minor values equal to one. In this particular business, the owner chooses to equal “major” with “branch”, and so when the second iBeacon is installed in the main branch, it gets a “minor” equal to two, but the major is still one.

Now, as soon as new branches install iBeacons, they will use the same UUID, but the “major” number will of course be different.

Meanwhile, throughout the city, other businesses roll their own iBeacons, and each business chooses to give different meanings to the “major” and “minor” numbers.

To summarise, each UUID is meant to be used on a single store, hospital, building or other location, and an application will listen typically to just one UUID. The “major” and “minor” numbers can be assigned any semantic that you see fit; branches and beacons, floors and rooms, seat row and number, it’s up to you to decide what they mean in your context.

In terms of distance, iBeacon sensing can be roughly categorised in three distance regions:

1. Immediate, up until half a meter;
2. Near, up until 2 meters;
3. Far, up to 30 meters.

Your own application should filter the values depending on the distance required by each use case, and present information to the user in a contextual way.

In terms of compatibility, currently iOS 7, Android 4.3, BlackBerry 10 and OS X 10.9 Mavericks ship with off-the-box support for BLE. In the case of iOS and OS X, this is both for emission and reception; any iOS device or OS X computer with the minimum requirements shown in the slide behind me can become iBeacons, enabling interesting scenarios in your own applications.

From a point of view of hardware, all iOS devices released since 2011 have BLE support built-in.

Some Android devices have support for detecting iBeacons as well,

the most important of which are enumerated in the slide behind me. By the way, there is an open source Java library on Github published by Radius Networks, for those of you who are into this kind of geek stuff, that simplifies the use of iBeacons in Android applications.

In the case of Blackberry there are several devices on the market that are already compatible with BLE, in particular the Q10, Z10, Z3, Z30 and the Porsche Design P'9982 (which is the one that Beyoncé uses, if you are interested in celebrities.) BlackBerry has published code on Github that shows how to consume iBeacons from your BlackBerry applications.

Windows Phone is a bit behind as we speak, because even if Windows Phone 8.1 has software support for BLE connections, it does not offer (yet) a public API, which means that at the moment WP developers cannot include support for iBeacons in their applications. Another problem is that currently the latest Nokia devices ship with Bluetooth 3.1, not with 4.0, which would prevent the software from working, anyway. The support for BLE has been announced for later this year.

But what about other platforms?

If you are using Xamarin and C# to create cross-platform apps, here's a bit of sample code on Github that will show you how. Please note that for the moment this code is only compatible with iOS and Android devices, but it should work with Windows Phone devices at some point in the future.

For those of you using PhoneGap to create cross-platform solutions, there is a project in Github that will surely interest you; an (at the moment) iOS-only plugin, where the developers are right now working on the Android compatibility layer, and which would help creating cross-platform solutions using this framework.

The same can be said about Appcelerator Titanium, here you have a module that will most certainly help you.

If you are a "low-level programming" kind of guy, you will be happy to know that you can use C and directly link to the "bluez" library for your apps. Which takes me to a command-line app for setting and testing your beacons, as well.

And let's not forget about Node.js; if you need to have your server-side app interact with beacons, here you go.

Oh, and of course you can use an Arduino or a Raspberry Pi to create your own iBeacons; which is exactly what I have done for tonight!

Finally, we have also iBeacon-enabled sunglasses. Which means that iBeacon-enabled contact lenses are just a couple of years away. Which means that iBeacon-enabled eye implants are just a decade or two away. Which means that Mr. Yakamoto will be greeted in GAP with blinking eyes sooner than you think.

And we've closed the circle.

Let’s talk now about iBeacon providers. Who builds iBeacons and how much do they cost? I’ve got here a quick panorama of providers to help you choose the best provider of iBeacon technology.

There are many companies shipping different kinds of iBeacons, these are the ones I found out, but there might be of course many more; in the table you can see the price ranges, and whether or not these providers display the “iBeacon Certified by Apple” logo in their website:

- Alibaba.com http://www.alibaba.com is of course worth a check in any case, as a simple search for “iBeacon” yields hundreds of different results. You might want to know that based on reports by colleagues and friends, the quality level of the beacons found through this channel varies considerably, and for example in many cases they feature low quality batteries and no security at all. So, be warned; you may get what you pay for.
- Appflare http://www.appflare.com
- Estimote http://estimote.com - they are probably one of the best-known providers of iBeacons right now.
- Glimworm Beacons http://glimwormbeacons.com
- IoT Design Shop http://www.iotdesignshop.com/beacon
- Kontakt.io http://kontakt.io
- Onyx Beacon http://www.onyxbeacon.com
- PassKit https://passkit.com/buy-ibeacon/
- Radius Networks http://www.radiusnetworks.com/ibeacon/
- Roximity http://roximity.com
- Stick & Find https://www.sticknfind.com/Beacons&iBeacons/
- Twocanoes Software http://twocanoes.com/bleu-station Some providers do not give information about pricing:
- BlueCats http://www.bluecats.com
- Gimbal (a Qualcomm company) https://www.gimbal.com
- JAALEE http://www.jaalee.com/beacon_en.html
- Rococo Software http://www.rococosoft.com
- shopkick http://www.shopkick.com/shopbeacon
- Swirl http://www.swirl.com/platform.html#beacons

As you can see, the price range varies considerably, from CHF 22 to CHF 52 per beacon, with an average of CHF 30. By ordering in bulk you can lower the price per beacon up to 16 or 17 CHF per beacon. Some companies do not advertise prices online, while others offer considerable discounts for bulk orders.

There are as well some marketing management platforms that take iBeacons into account, here mentioning:

- Adobe Marketing Cloud
- Appflare Cloud
- inMarket
- Passjoy
- PassMarket
- Pushmote

- Roximity Platform
- Swirl

I would like to point out that Trifork does not endorse any of these brands nor guarantee their operations or products, and this information is offered here as a guideline and an orientation.

Usability and User Experience

Now, let's move to the third part of our talk of tonight, about Usability and User Experience. We all know that Apple is all about the "user experience" of their products, and of course iBeacon is not an exception; the whole BLE vs. NFC debate was cut short internally at Apple after the UX designers realised the potential for immersive experiences thanks to the characteristics offered by BLE. In terms of usability, the first major issue with iBeacons is the multiplicity of applications that users will be required to have in their devices in order to take advantage to them. This means three things:

1. iBeacon is right now the wild west. First to arrive grabs it all.
2. There is a market window of opportunity for aggregators, like malls and shopping centers, to centralize beacon information in their own applications.
3. Users will start dismissing apps that are simply too annoying in terms of notifications and activations, and only those apps that are not pushy, irrelevant and that really provide value will win.

By the way, even if the user does not delete the app in case of annoying behavior, remember that there is a "master switch" for iBeacon: users are able to block its access to location information; doing so will actually disable any iBeacon functionality in your apps, so you'd better do things right to begin with.

Another important point to consider is that there are actually two ways iOS devices can consume and interact with iBeacons:

1. The first is Apps, of course; an app can be programmed to "sniff" for nearby iBeacons, and to change its own behaviour depending on the closest beacon.
2. But there is another means, which does not require an ad-hoc app, it is much cheaper to implement and use, and one that integrates wonderfully well with iOS devices and users; Passbook. Indeed, passbook items such as tickets, coupons, boarding passes and others can be "iBeacon-aware" and they can automatically appear in the home screen of the user when they are close to a particular beacon.

We are going to see both interactions in the demos right after.

Finally, the most important thing to remember: an iBeacon is not just a glorified QR Code, ok? They are much, much more; first of all they do not require any user intervention (a priori at least) and they are

much more discrete visually. Even better, using Passbook, QR Codes and iBeacon can work together seamlessly.

Some best QR Code practices apply to iBeacons, for example:

- Make sure iBeacons serve a real purpose
- Pay attention to the placement of the beacons
- Disclose their benefit to users
- Test them!
- Track them!

As much as possible, we should avoid with iBeacons the reaction we have had with QR codes, particularly after some failed attempts to use them properly.

Security and Privacy

Finally, the last part of our talk, Security and Privacy. I am first going to tackle a little threat management here, just disclosing some issues that can arise when working with iBeacons:

- Hijacking iBeacon username and password
- Stealing UUIDs and spoofing users
- Battery drain
- Management
- Connectivity / Compatibility
- Cost (Theft, insurance, etc)

In terms of privacy issues, you have to keep in mind that iBeacon are the real life equivalent of a browser cookie. You know, browser cookies? Those little pesky files that help marketing teams to track you while you browse around the web?

Just as with browser cookies, iBeacons also generate their own privacy issues; what happens with the data that you will gather through your application?

Whenever you go to a website like Twitter, you see a disclaimer that explains users that you are using cookies to track them; should we have similar banners in our shops now?

The most important thing to remember now is that we should apply all the knowledge we have from previous “live marketing” experiences, such as QR codes, privacy scandals and other issues, to actually avoid them now with iBeacons. In particular, brace for the first scandal that will arise after some company misuses this technology, because I think it could happen really fast.

Case Studies

Here a small, quick list with major iBeacon implementations from around the world:

- Apple Stores in the US, to help the customer find their products on the stores.
- Virgin Atlantic (with Estimote) for Passbook coupons for free items.
- MacWorld / iWorld, for a scavenger hunt with prizes (with Twocanoes)
- Major League Baseball (MLB) for helping users find their way in stadiums.
- National Basketball Association (NBA)
- Macy's (with shopkick)
- Consumer Electronics Show (CES)
- United Nations Museum in NYC, where they simulate a minefield to raise awareness on the personal mine problem.
- SXSW Festival for attendee check in and badge pickup, and interactive sessions.
- BeHere app: track students attendance in classrooms.

For an updated list of iBeacon use cases, be sure to check this page[1].

Conclusion

So, what is the future of all this?

As always with Apple, speculation is a dangerous game. Everybody knew that Apple was going to release a cell phone, but clearly nobody expected the iPhone, the App Store and later the iPad to change the way we interact and consume information in such a profound way.

It is very tempting to speculate with payments as the next possible step for Apple. Indeed, iBeacon-powered apps can detect their presence near to an iBeacon-emitter point-of-sale application, triggering a dialog between both devices and making the payment process as easy as with NFC. Actually PayPal is moving in this direction too, and they are pushing very hard their own standard for BLE beacons... so there will be lots of movement in this area in the next few months.

And, by the way, if this payments rumour turns out to be true, believe me, you will be happy to have some Apple stock in your portfolio. But then again, you haven't heard this from me or Trifork at all!

In any case the important thing for me now is to give you the promised four answers to the questions asked at the beginning of this presentation; here they go:

1. Marketing —> "Minority Report" & Analytics
2. Technology —> Bluetooth LE & "Lighthouse"
3. Usability & UX —> Apps & e-Wallets
4. Security & Privacy —> Cookie & Annoyance

[1] http://www.mobisfera.com/summary-real-cases-close-real-using-ibeacons/?lang=en

Ten Years

Adrian Kosmaczewski

2014-11-06

I’ve been blogging for 10 years now. From Movable Type to Wordpress.

I started blogging literally days before I met Claudia.

I traced my job, my masters degree, my business, my private life, and now my conversion into poetry and literature.

Several hiatus: in 2006, and later in 2012/2013; probably the most complex and difficult time of my recent life.

Now writing is not only about technology; it is mostly a life or death kind of situation, where I can unload feelings from my heart and pour them outside of my soul. Most of what I write lately is cryptic for most of you; I know it. Just read it as it is, without judgement, feeling the words. You will understand.

Cocoa is the new Carbon: the Future of Apple's Beloved Framework

Adrian Kosmaczewski

2015-02-26

In this talk, Adrian will provide lots of speculation and highly arguable unverified gossip, about how the design of Swift will lead Apple to redesign Cocoa into new directions, and maybe replacing it altogether. Some code examples will help him defend a point that nobody outside of Cupertino can sustain for sure.

Introduction

A wise man called Heraclitus said 2500 years ago that "Change is the only constant." He said it in greek, obviously, so actually he said these words: "πάντα χωρεῖ καὶ οὐδὲν μένει." But I just do not know how to pronounce this, so I'll leave it there.

The point is that Apple loves change. And to begin my talk I will go quickly back in time to 1976, to 1984, to 1995, to the two thousands, and back to now. In the meantime I will buy as much Apple stock as I can. You never know.

Look at the progression. We've gone from very humble machines to state-of-the-art computers, some of them handheld and operating a social and technological revolution. Soon, watches are going to appear in the horizon to complete this lineup, too.

Look at their CPUs; Apple has gone from a humble MOS 6502 to Gigaherz-fast ARM chips, eating very little power and providing enough calculation power to put a man on the moon, or, you know, to publish a selfie in Instagram.

At the same time, we have evolved from 8 bit to 64 bit architectures, forcing us to rewrite lots of low-level pointer-arithmetic code, because you know, that's where the fun is.

And finally, the most important part of the evolution for us, software developers, is the constant change of "official" programming languages; from Wozniak's Integer BASIC in 77 to Swift, Apple has constantly required us to migrate our skills to newer, safer, and buggier programming languages.

This leads me to another clear evolutionary path, if you look at this slide closely: we have gone from a procedural world, the one of BASIC and Pascal, to an object-oriented one, with C++ and Objective-C, and Swift is opening the way for a more functional way of doing things.

Of course, every time the marketing department said the same thing about those languages; that they are the best, the fastest, the blah blah blah. So, in a sense, we can say that "plus ça change, plus c'est la même chose." Clearly, Jean-Baptiste and Heraclitus never agreed on the subject of change.

Transitional Technologies

Another thing that has not changed is that Apple always provided us with what I call "Transitional Technologies." That is, emulators, compatibility layers and binary formats that have allowed us to move our applications from one architecture or language to the next, and often to run it without changes in the new platform. The history of Apple is quite interesting in this respect.

The earliest example is the "Mac 68k Emulator Layer," bundled with Power Macs back in the 90's, so that old Mac applications compiled for the Motorola 68000 CPUs could run. Remember that? This layer existed even in Classic, yet another compatibility layer that allowed OS 9 applications to run on OS X… but I digress.

Then we have the Macintosh Application Environment. Anyone remember this one? Back in the days where we were waiting for Jobs to return, when Copland, Taligent and OpenDoc were still worthy of attention, back in those days Apple released a compatibility layer for Sun OS Unix boxes with SPARC chips, where Mac apps could run. I guess I do not need to say why nobody remembers it.

More recently, some of us remember the "Classic Environment," allowing users to run their old OS 9 apps in Cheetah, Jaguar and Tiger.

Going back to the compatibility layers, in the 2000's Apple introduced "Carbon." It was a sibling framework to Cocoa, deprecated in Mountain Lion, never ported to 64 bits, allowing OS 9 applications to be recompiled as native OS X apps, even if the look and feel was not exactly the same, but at least you would get Aqua buttons, which is everything users wanted back in 2002.

Carbon was Apple's transitional API, the one that enabled OS 9 applications written in C and C++ to be compiled as native OS X applications, waiting for developers to rewrite these applications as native Cocoa apps in Objective-C. Something that the Photoshop team took ages to do, but well, that's another story.

In the meantime, preparing for even more changes, Apple started two famous very open source projects; the first is WebKit, and I think everybody agrees on how important it has been for the web. But more important for us tonight is LLVM, a project that initially started as a

way to modernise the old and rusty GCC compiler infrastructure, and Swift can be said a direct descendant of LLVM.

More or less by that time, Apple decided to use Intel chips instead of PowerPC ones, and so it introduced two transitional technologies at the same time: first "Rosetta," an emulator layer which allowed OS X PowerPC apps to run on Intel chips, until their developers would recompile them as native i386 ones. After recompiling them, for a while Xcode would produce "Universal Binaries." They are basically NeXT fat binaries, merged with the "lipo" tool, containing both PowerPC and Intel code; applications built using this technology could run seamlessly in older and newer Macs, between 2006 and 2011.

Based on the new LLVM compiler infrastructure I mentioned before, Objective-C evolved for the first time since 1988, and in 2006 Apple released Objective-C 2.0, which brought many new features:

- Garbage collection
- Properties
- Fast enumeration
- 64-bit capabilities

LLVM is also the original technology that brought us ARC or the Xcode Static Analyzer; in particular, ARC is a fundamental piece for Swift, providing compile-time memory management, instead of relying on a potentially heavy garbage collected runtime.

Cocoa

Carbon was introduced in 2000, and deprecated in 2012, so, my question is, maybe Cocoa is the new Carbon? Is Cocoa the new transitional API that will enable Objective-C applications to run in OS X and iOS while developers rewrite them in Swift, while we wait for Apple to release the… what? The Swift "Birdy" Framework?

I argue that we are back again in a transitional period. Objective-C will most probably only be updated as long as Swift requires it; the new "nullable" pointer annotations introduced recently are a clear indication of that. Objective-C is evolving into a language that supports future Swift libraries. New hardware like the Watch will come and will require new software.

My theory is that Cocoa will be around for the next 10 years. That is right; Cocoa and Objective-C will be around until 2025, and during this time, of course, Swift will become the de facto programming language for writing Mac, iOS and Watch applications. And the demise of Objective-C will surely make some developers very happy.

The scenario is all set for yet another migration. And Cocoa is the compatibility layer this time.

Inspiration

With all of this historic background, let's take a closer look at Swift. What is Swift? Apple says that the language can be defined as "Modern, Safe, Fast and Interoperable." For the moment I can say that Swift is a great language to create strongly-typed "Hello World" applications. But I'm probably too grumpy, so don't take my word for it.

One of the best explanations comes from Twitter: "Swift is Haskell with C# syntax." In my personal experience with the language, beyond cursing the pitiful state of the developer tools, I came to the conclusion that Swift is a hybrid language compatible with both object-oriented and functional APIs. That is what I call Swift. You might disagree with this definition, and we could talk it over a beer after this talk, but in essence this is how I see the language right now.

Chris Lattner, the creator of Swift, says that one of the foundational bricks of the language are protocols; he has not really mentioned the functional part as one of those bricks, yet the developer community has a different opinion.

All of this does not tell us much about that "Birdy" framework of 2019; at best, I think we can just grab some inspiration from other languages and their associated frameworks. What features of the language would influence the design of this master framework?

- Protocols
- Strong typing
- Optionals
- Generics
- Tuples
- Enums

Furthermore, what could we borrow from other libraries or frameworks, to imagine the next version of Cocoa? Let's review quickly some popular programming languages, and see what could be borrowed from each.

From C

I think that LINQ is a great example of something that could and probably should be ported to Swift.

Oh, and by the way, maybe Xcode could take a hint or two from Visual Studio, if you ask me.

From JavaScript

There are great examples of short, sweet JavaScript libraries around, most of them heavily developed during the past 10 years, after the explosion of AJAX. For example jQuery, maybe the quintessential

JavaScript framework; small, it bridges compatibility problems between browsers, it uses JavaScript's functional features extensively, and it has inspired a whole industry of JavaScript frameworks such as Zepto, Sencha Touch, YUI, Ext.js, Underscore, and the list goes on.

Actually I do expect somebody to come up with a Node.js-like web server written in Swift; it's only a matter of time.

From Haskell and Erlang

What should we take from these? Anything but the syntax. I think we got that, as a matter of fact; so the key elements we could adapt are actors, parallel execution and fail-safe features, many of which are already available in the language, to be fair.

From Ruby and Rails, RSpec, Cucumber…

We can borrow the great DSLs built on top of it; and I look in particular to CocoaPods, which is simply a DSL based in Ruby that help us handle external dependencies in our projects. Could not we just use Swift to do the same? I think we could and we should.

The fact that Swift allows "last parameter" lambdas to be called without parenthesis is very similar to the mechanism enabled by the "yield" keyword in Ruby.

From C++

One could argue that the current Swift standard library borrows some functions from the C++ standard library, and maybe also a bit from Boost. Although generics are not quite the same thing as C++ templates, in spite of the similar syntax, we can use them to compose objects in very interesting directions.

I would like to see who will be the Alexandrescu of Swift, providing us with a full book of crazy uses of generics, redefining what we can do with them, including enums, structs and not only classes in the mix.

From Java and Scala

We could take IntelliJ's AppCode (thankfully!) and then lots of examples of how not to do things: Eclipse, Maven, Struts, the low speed of compilation and execution, and I'm pretty sure you have plenty of other examples.

Let's not repeat those mistakes, please.

From Scala, of course, we have an uncannily similar syntax, one that stops there, of course, because the execution models of both languages are quite different, as a matter of fact.

Gedankenexperiment

So, the question is, what would Apple's official Swift framework include? What would it look like?

I will cheat; in this talk I will see the current state of Open Source development in Swift to provide us with some hints. Maybe Apple is also paying attention to these projects, and will give us APIs looking and behaving like the ones I'm going to describe now.

I was going to show some live coding demos, but frankly the current state of playgrounds and the Swift compiler in general played against me tonight. So I will stick with very simple code samples right on the slides.

Swift Standard Library

Of course the first example to pay attention to is Swift's own standard library. I cannot avoid looking at it and thinking that it is not yet ready; that it is essentially incomplete, yet it is probably the embryo of the next evolution of Cocoa.

The most important observation I can make from typical code examples using the standard library is how few classes you see in use; most of the APIs are functions operating on structs; even better, these structs can be made immutable, with all the benefits that brings.

So, few classes. You know a Swift developer has issues when you spot a static class method somewhere during a code review.

On the negative side, one thing that the Standard Library needs, and fast, is a more complete documentation of it, provided and maintained from Apple. It is actually embarrassing that we have to rely on SwiftDoc[1] instead.

LINQ in Swift

I have found a complete project in Github exploring the possibility of using Swift's standard library in a similar fashion to LINQ in C#. The slide shows a snippet of code in C#.

But to be fair, being able to query data structures in memory using a SQL-like language is something that Cocoa has been able to do since NSPredicate was introduced, back in the early 2000, together with Core Data; NSPredicate itself is heavily dependent upon KVO, another technology available since the 80's in Cocoa.

However, neither NSPredicate nor KVO provide the compile-time verification power of C#, not even remotely. Could we use Swift, which comes bundled with such power, to evolve those technologies into a LINQ-like library? I think that the answer is a resounding yes.

[1] http://swiftdoc.org

Dollar.swift

One Swift library that has been heavily inspired by jQuery and Underscore is “Dollar.swift” for example. It provides several functional constructions, allowing developers to operate directly on functions, compose and reuse them as required. This library makes heavy use of everything functional that Swift has to offer.

Of course, Swift and JavaScript have dramatically different runtime differences; JavaScript runs in a single-threaded environment, and for performance reasons, JavaScript frameworks tend to be asynchronous, delegating behaviour to “timeout’d” functions, in order not to block the browser’s run loop.

But just as JavaScript, Swift is a hybrid, “kinda-functional” language, with a syntax that allows for similar constructions. Let’s imagine for a second a library such as Socket.io, but written in Swift? The mind boggles.

Alamofire for networking

This is probably one of the most interesting libraries available for Swift today, and I’m pretty sure that everyone will use it. It has very interesting characteristics, showing how to build asynchronous APIs in Swift.

Look at this code; we have a very “JavaScript” like syntax. We specify the request that we want to send to the server by chaining a couple of functions after the others. The parameters are simple, and some anonymous functions are used as asynchronous callback methods.

In Swift, methods are actually curried functions; not only that, but functions map directly to Objective-C blocks, and as such they can be used in the same situations, with the same idioms.

NSNotifications for Architecture

There are few components so loved and misused in Cocoa like the NSNotificationCenter. We all love it and hate it vehemently at the same time. So it is not a surprise that both Mike Ash and Chris Eidhof have both thought of replacing it with a more “Swift-like” API, and in this case I will present the one shown by Chris.

As you all know, and if you don’t now you do, Chris is one of the creators of objc.io, the greatest online magazine about Cocoa there ever was. He is also one of the co-authors of the book “Functional Programming in Swift” so I guess he knows what he’s talking about.

So that’s why one day he presented a replacement for the Notification Center, and it looks like this.

- Generics for notification definitions
- Lambdas as callbacks

Swift Events for communication

Events are quintessential in UI toolkits; they allow developers to receive asynchronous information about the activities of users. But an event is essentially an object that wraps some code that has to be executed at some point in the future; so a very simple, almost naïve implementation of events could be the one shown in this slide.

To use this API, just assign some handlers to an event, and use the `raise` method to call them all at once. Simple.

Pay attention to a very interesting feature of Swift! Methods that take multiple parameters are the same as methods that take tuples as input. This makes the syntax used to raise events much simpler, straightforward and easier to read.

Swifty User Defaults for storage

This library is a perfect example of small additions on top of NSUserDefaults, in order to have a more "Swift-like" API. The author uses subscripts to enable a simpler syntax for getters and setters using subscripts, as well as cleverly using generics to provide a strongly-typed interface.

Swift Moment for time manipulation

Finally, without shame, I'd like to talk about my own take on a pure Swifty implementation of a date & time library. Swift Moment was recently featured by Natasha Murashev and Dave Verwer in their respective newsletters, and it has been heavily inspired by a JavaScript library called Moment.js, quite popular in web development circles.

Swift Moment is based in several typical Swift paradigms.

- The most important being the heavy use of functions and structs instead of classes and methods. I have made the early choice of making the library as similar as possible to Moment.js, and as such I use global functions, such as "moment()" to create new instances of a "Moment" struct. This structure encapsulates an NSDate instance, and it provides an easier API to typical date and time operations, such as conversion to strings, getting components such as day of the week or others, and many other operations.
- Another thing that I've extensively used in Swift Moment are operators. Being able to say whether a date is older than another, or similar, are cases where operators work beautifully well.
- Finally, I've tried as much as possible to make immutable structs out of Moment and Duration. It makes sense; they represent fixed points in time, or fixed time spans, and modifying them at runtime makes no sense.

When using Swift Moment, developers have a straighforward interface. Ambiguity is almost nonexistent.

Conclusion

I cannot claim, by any means, to know what is going on inside of Cupertino as we speak. I think, however, that Swift will move the development of Cocoa into new directions, thanks to the many features of Swift:

- Protocols, even on enums and structs
- Pattern Matching
- Functions chaining
- Function currying
- Subscripts
- Functions and structs instead of classes and methods.
- Custom operators, where it make sense.
- Immutability
- Optionals
- Tuples
- Generics

See the pattern? There are not so many objects or classes. APIs are strongly typed. Optionals make sense. Data is immutable. Function composition takes over object composition. Pattern matching and tuples here and there. Lambdas passed around as first-class citizens. Swift has a tremendous potential in terms of syntax and expressive power, and I frankly think that we have not seen anything yet.

On the minus side, I have to say that I would like to see more reflection capabilities in Swift; it is something that C# and .NET offer, and heck, even Objective-C had it, and we've lost it. Powerful frameworks have reflection.

In any case, I think that we are moving towards a functional world, and as such I recommend everyone – including myself – to learn more about the Functional Programming Paradigm, in order to be a better post-Cocoa developer, thanks to Swift. And Junior will talk about that right after this talk.

However, it is not very clear to me what will be the next step regarding user interfaces. We all know that there is a tension between UI development and functional programming, because the former is all about state, and the latter frowns upon it. Will we see a functional-based UI component library? Does this make any sense? Or will UIKit (and maybe UXKit) simply change to support Swift paradigms?

I guess, time will tell. Thanks a lot for your attention.

Special thanks to Fabrice Truillot, who has been writing Mac applications for 27 years in many of the languages I mentioned, for reviewing drafts of this speech and providing great feedback.

Being A Developer After 40

Adrian Kosmaczewski

2016-04-25

This is the talk I gave at App Builders Switzerland[1] on April 25th, 2016.

The slides are available on SpeakerDeck[2] and at the bottom of this article. The video of the session is available in YouTube[3]. This article has been printed in the June 2016 edition of Hacker Bits[4].

Thanks to some awesome translators, here are some localized versions of this article:

- русский[5]
- Magyar[6]
- Čeština[7]
- Português[8]
- Persian — فارسی[9]
- Vietnamese — Tiếng Việt[10]
- Românesc[11]

Hi everyone, I am a forty-two years old self-taught developer, and this is my story.

A couple of weeks ago I came by the tweet below, and it made me think about my career, and those thoughts brought me back to where it all began for me:

---

[1] https://www.appbuilders.ch

[2] https://speakerdeck.com/akosma/being-a-developer-after-40

[3] https://www.youtube.com/watch?v=GQx_beRMHVg

[4] http://hackerbits.com/adrian-kosmaczewski-june-2016/

[5] https://habrahabr.ru/post/282674/

[6] http://m.logout.hu/cikk/fejlesztonek_lenni_40_felett/fejlesztonek_lenni_40_felett.html

[7] http://blog.zvestov.cz/item/176

[8] https://web.archive.org/web/20200215093736/https://lfbittencourt.com/sendo-desenvolvedor-depois-dos-40-db274feb9445/

[9] https://virgool.io/@abbasmousavi/%D8%A8%D8%B1%D9%86%D8%A7%D9%85%D9%87%E2%80%8C%D9%86%D9%88%DB%8C%D8%B3-%D8%A8%D9%88%D8%AF%D9%86-%D8%A8%D8%B9%D8%AF-%D8%A7%D8%B2-%DA%86%D9%87%D9%84-%D8%B3%D8%A7%D9%84%DA%AF%DB%8C-being-a-developer-after-40-hwgu50jp5ih9

[10] https://techmaster.vn/posts/33882/lap-trinh-vien-lon-tuoi

[11] https://rolandleth.com/a-fi-programator-dupa-40-de-ani

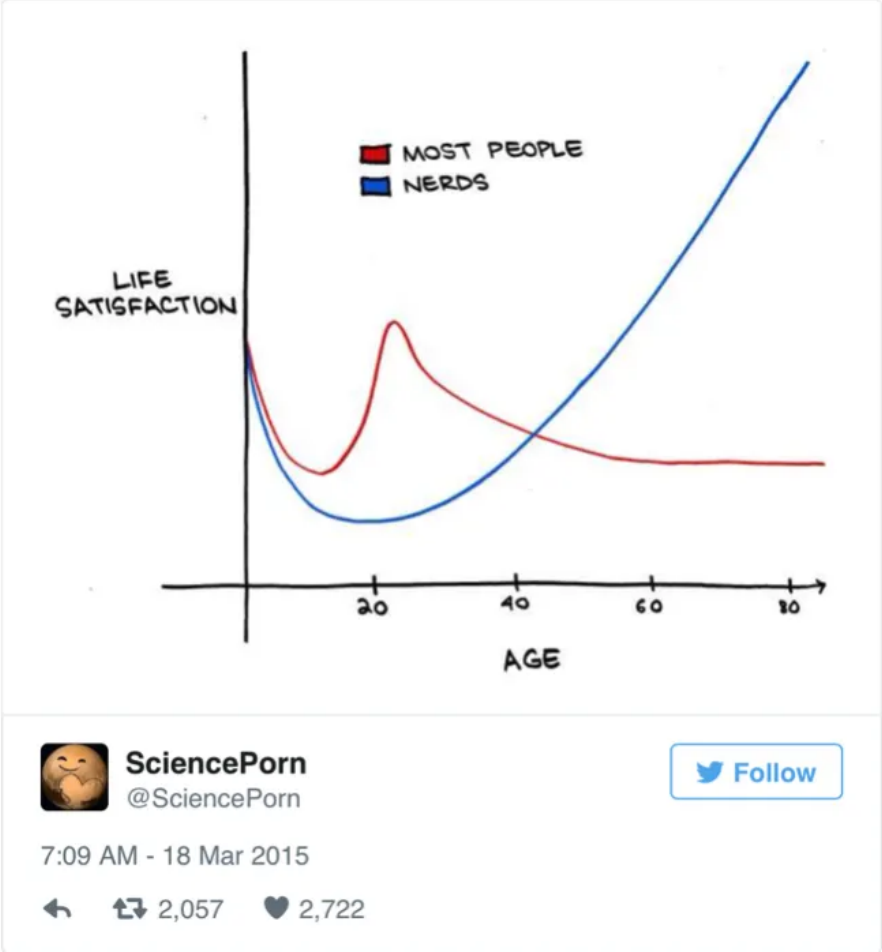


Figure 1: Original: http://www.smbc-comics.com/index.php?db=comics&id=2436

I started my career as a software developer at precisely 10am, on Monday October 6th, 1997, somewhere in the city of Olivos[12], just north of Buenos Aires[13], Argentina[14]. The moment was Unix Epoch 876142800[15]. I had recently celebrated my 24th birthday.

The World In 1997

The world was a slightly different place back then.

Websites did not have cookie warnings. The future of the web were portals like Excite.com[16]. AltaVista was my preferred search engine. My e-mail was kosmacze@sc2a.unige.ch, which meant that my first personal website was located in http://sc2a.unige.ch/~kosmacze. We were still mourning Princess Lady Diana[17]. Steve Jobs had taken the role of CEO and convinced Microsoft to inject 150 million dollars[18] into Apple Computer. Digital Equipment Corporation was suing Dell. The remains of Che Guevara had just been brought back to Cuba. The fourth season of "Friends"[19] had just started. Gianni Versace[20] had just been murdered in front of his house. Mother Teresa[21], Roy Lichtenstein[22] and Jeanne Calment[23] (the world's oldest person ever) had just passed away. People were playing Final Fantasy 7[24] on their PlayStation[25] like crazy. BBC 2 started broadcasting the Teletubbies[26]. James Cameron was about to release Titanic[27]. The Verve had just released their hit "Bitter Sweet Symphony"[28] and then had to pay most royalties to the Rolling Stones.

Smartphones looked like the Nokia 9000 Communicator[29]; they had 8 MB of memory, a 24 MHz i386 CPU and run the GEOS operating system.

Smartwatches looked like the CASIO G-SHOCK DW-9100BJ[30]. Not as many apps but the battery life was much longer.

---

[12] https://en.wikipedia.org/wiki/Olivos,_Buenos_Aires
[13] https://en.wikipedia.org/wiki/Buenos_Aires
[14] https://en.wikipedia.org/wiki/Argentina
[15] http://www.epochconverter.com
[16] http://excite.com
[17] https://en.wikipedia.org/wiki/Death_of_Diana,_Princess_of_Wales
[18] http://www.cnet.com/news/microsoft-to-invest-150-million-in-apple/#!
[19] https://en.wikipedia.org/wiki/Friends_%28season_4%29
[20] https://en.wikipedia.org/wiki/Gianni_Versace
[21] https://en.wikipedia.org/wiki/Mother_Teresa
[22] https://en.wikipedia.org/wiki/Roy_Lichtenstein
[23] https://en.wikipedia.org/wiki/Jeanne_Calment
[24] https://en.wikipedia.org/wiki/Final_Fantasy_VII
[25] https://en.wikipedia.org/wiki/PlayStation_%28console%29
[26] http://www.forbes.com/sites/oracle/2015/07/31/prepare-for-the-teletubby-trained-mobile-workforce/#75086f3b7fb3
[27] http://www.imdb.com/title/tt0120338/
[28] https://www.youtube.com/watch?v=1lyu1KKwC74
[29] https://en.m.wikipedia.org/wiki/Nokia_9000_Communicator
[30] http://www.g-shock.eu/euro/aboutgshock/history/

Figure 2: Excite in 1997, courtesy of the Internet Archive

IBM Deep Blue had defeated for the first time[31] Garry Kasparov in a game of chess.

A hacker known as "_eci" published the C code for a Windows 3.1, 95 and NT exploit called "WinNuke,"[32] a denial-of-service attack that on TCP port 139 (NetBIOS) causing a Blue Screen of Death.

Incidentally, 1997 is also the year Malala Yousafzai[33], Chloë Grace Moretz[34] and Kylie Jenner[35] were born.

Many film storylines take place in 1997, to name a few: Escape from New York[36], Predator 2[37], The Curious Case of Benjamin Button[38], Harry Potter and the Half-Blood Prince[39], The Godfather III[40] and according to Terminator 2: Judgement Day[41], Skynet became self-aware at 2:14 am on August 29, 1997. That did not happen; however, in an interesting turn of events, the domain google.com had been registered on September 15th that year.

We were two years away from Y2K[42] and the media were starting to get people nervous about it.

My First Developer Job

My first job consisted of writing ASP pages in various editors, ranging from Microsoft FrontPage[43], to HotMeTaL Pro[44] to EditPlus[45], managing cross-browser compatibility between Netscape Navigator and Internet Explorer 4, and writing stored procedures in SQL Server 6.5[46] powering a commercial website published in Japanese, Russian, English and Spanish — without any consistent UTF-8 support[47] across the software stack.

The product of these efforts ran in a Pentium II[48] server hosted somewhere in the USA, with a stunning 2 GB hard disk drive and a whooping 256 MB of RAM. It was a single server running Windows NT 4[49], SQL

[31] https://en.wikipedia.org/wiki/Deep_Blue_versus_Garry_Kasparov
[32] https://en.wikipedia.org/wiki/WinNuke
[33] https://www.malala.org/malalas-story
[34] http://www.imdb.com/name/nm1631269/
[35] https://en.wikipedia.org/wiki/Kylie_Jenner
[36] http://www.imdb.com/title/tt0082340/
[37] http://www.imdb.com/title/tt0100403/
[38] http://www.imdb.com/title/tt0421715/
[39] http://www.imdb.com/title/tt0417741/
[40] http://www.imdb.com/title/tt0099674/
[41] http://www.imdb.com/title/tt0103064/
[42] https://en.wikipedia.org/wiki/Year_2000_problem
[43] https://en.wikipedia.org/wiki/Microsoft_FrontPage
[44] http://www.hotmetalpro.com/press/
[45] https://www.editplus.com
[46] http://stackoverflow.com/questions/194/upgrading-sql-server-6-5
[47] http://www.alanwood.net/unicode/netscape.html
[48] https://en.wikipedia.org/wiki/Pentium_II
[49] https://en.wikipedia.org/wiki/Windows_NT_4.0

Server 6.5[50] and IIS 2.0[51], serving around ten thousand visitors per day.

My first professional programming language was this mutant called VBScript[52], and of course a little bit of JavaScript on the client side, sprinkled with lots of "if this is Netscape do this, else do that" because back then I had no idea how to use JavaScript properly.

Interestingly, it's 2016 and we are barely starting[53] to understand how to do anything in JavaScript.

Unit tests were unheard of. The Agile Manifesto[54] had not been written yet. Continuous integration was a dream. XML was not even a buzzword. Our QA strategy consisted of restarting the server once a week, because otherwise it would crash randomly. We developed our own COM+[55] component in Visual J++[56] to parse JPEG files uploaded to the server. As soon as JPEG 2000[57]-encoded files started popping up, our component broke miserably.

We did not use source control, not even CVS[58], RCS[59] or, God forbid, SourceSafe[60]. Subversion[61] did not exist yet. Our Joel Test[62] score was minus 25.

6776 Days

For the past 6776 days I have had a cup of coffee in the morning and wrote code with things named VBScript, JavaScript, Linux, SQL, HTML, Makefiles, Node.js, CSS, XML, .NET, YAML, Podfiles, JSON, Markdown, PHP, Windows, Doxygen, C#, Visual Basic, Visual Basic.NET, Java, Socket.io, Ruby, unit tests, Python, shell scripts, C++, Objective-C, batch files, and lately Swift.

In those 6776 days lots of things happened; most importantly, my wife and I got married. I quit 6 jobs and I was fired twice. I started and closed my own business. I finished my Master Degree. I published a few open source projects, and one of them landed me an article on Ars Technica by Erica Sadun herself[63]. I was featured in Swiss and Bolivian TV shows. I watched live keynotes by Bill Gates and by Steve Jobs in

[50] https://en.wikipedia.org/wiki/Microsoft_SQL_Server
[51] https://en.wikipedia.org/wiki/Internet_Information_Services
[52] https://en.wikipedia.org/wiki/VBScript
[53] http://www.planningforaliens.com/blog/2016/04/11/why-js-development-is-crazy/
[54] http://www.agilemanifesto.org
[55] https://en.wikipedia.org/wiki/Component_Object_Model#COM+
[56] https://en.wikipedia.org/wiki/Visual_J%2B%2B
[57] https://en.wikipedia.org/wiki/JPEG_2000
[58] http://www.nongnu.org/cvs/
[59] https://www.gnu.org/software/rcs/
[60] https://en.m.wikipedia.org/wiki/Microsoft_Visual_SourceSafe
[61] http://subversion.apache.org/
[62] http://www.joelonsoftware.com/articles/fog0000000043.html
[63] http://arstechnica.com/apple/2009/04/iphone-dev-convert-xib-files-to-objective-c/

Seattle and San Francisco. I spoke at[64] and co-organised conferences in four continents. I wrote and published two books[65]. I burned out twice (not the books, myself,) and lots of other things happened, both wonderful and horrible.

I have often pondered about leaving the profession altogether. But somehow, code always calls me back after a while. I like to write apps, systems, software. To avoid burning out, I have had to develop strategies.

In this talk I will give you my secrets, so that you too can reach the glorious age of 40 as an experienced developer, willing to continue in this profession.

Advice For The Young At Heart

Some simple tips to reach the glorious age of 40 as a happy software developer.

1. Forget The Hype

The first advice I can give you all is, do not pay attention to hype. Every year there is a new programming language, framework, library, pattern, component architecture or paradigm that takes the blogosphere by storm. People get crazy about it. Conferences are given. Books are written. Gartner hype cycles[66] rise and fall. Consultants charge insane amounts of money to teach, deploy or otherwise fuckup the lives of people in this industry. The press will support these horrors and will make you feel guilty if you do not pay attention to them.

In 1997 it was CORBA[67] & RUP[68].

In 2000 it was SOAP[69] & XML.

In 2003 it was Model Driven Architecture[70] and Software Factories[71].

In 2006 it was Semantic Web[72] and OLPC[73].

In 2009 it was Augmented Reality[74].

In 2012 it was Big Data[75].

[64] http://www.infoq.com/presentations/Introduction-to-iOS-Software-Development
[65] http://amazon.com/author/akosma
[66] http://www.gartner.com/technology/research/methodologies/hype-cycle.jsp
[67] https://en.wikipedia.org/wiki/Common_Object_Request_Broker_Architecture
[68] https://en.m.wikipedia.org/wiki/Rational_Unified_Process
[69] https://en.wikipedia.org/wiki/SOAP
[70] https://en.wikipedia.org/wiki/Model-driven_architecture
[71] https://en.m.wikipedia.org/wiki/Software_factory
[72] https://en.wikipedia.org/wiki/Semantic_Web
[73] https://en.m.wikipedia.org/wiki/One_Laptop_per_Child
[74] https://en.m.wikipedia.org/wiki/Augmented_reality
[75] https://en.m.wikipedia.org/wiki/Big_data

In 2015… Virtual Reality? Bots?

Do not worry about hype. Keep doing your thing, keep learning what you were learning, and move on. Pay attention to it only if you have a genuine interest, or if you feel that it could bring you some benefit in the medium or long run.

The reason for this lies in the fact that, as the Romans said in the past, **nihil sub sole novum.** Most of what you see and learn in computer science has been around for decades, and this fact is purposedly hidden beneath piles of marketing, books, blog posts and questions on Stack Overflow. Every new architecture is just a reimagination and a readaptation of an idea that was floating around for decades.

2. Choose Your Galaxy Wisely

In our industry, every technology generates what I call a "galaxy." These galaxies feature stars but also black holes; meteoric changes that fade in the night, many planets, only a tiny fraction of which harbour some kind of life, and lots of cosmic dust and dark matter.

Examples of galaxies are, for example, .NET, Cocoa, Node.js, PHP, Emacs, SAP, etc. Each of these features evangelists, developers, bloggers, podcasts, conferences, books, training courses, consulting services, and inclusion problems. Galaxies are built on the assumption that their underlying technology is the answer to all problems. Each galaxy, thus, is based in a wrong assumption.

The developers from those different galaxies embody the prototypical attitudes that have brought that technology to life. They adhere to the ideas, and will enthusiatically wear the t-shirts and evangelize others about the merits of their choice.

Actually, I use the term "galaxy" to avoid the slightly more appropriate if not less controversial term "religion," which might describe this phenomenon better.

In my personal case, I spent the first ten years of my career in the Microsoft galaxy, and the following nine in the Apple galaxy.

I dare say, one of the biggest reasons why I changed galaxies was Steve Ballmer. I got tired of the general attitude of the Microsoft galaxy people against open source software.

On the other hand, I also have to say that the Apple galaxy is a delightful place, full of artists and musicians and writers who, by chance or ill luck, happen to write software as well.

I attended conferences in the Microsoft galaxy, like the Barcelona TechEd 2003, or various Tech Talks in Buenos Aires, Geneva or London. I even spoke at the Microsoft DevDays 2006 in Geneva. The general attitude of developers in the Microsoft galaxy is unfriendly, "corporate" and bound in secrecy, NDAs and cumbersome IT processes.

The Apple galaxy was to me, back in 2006, exactly the opposite; it was full of people who were musicians, artists, painters; they would write software to support their passion, and they would write software with passion. It made all the difference, and to this day, I still enjoy tremendously this galaxy, the one we are in, right now, and that has brought us all together.

And then the iPhone came out, and the rest is history.

So my recommendation to you is: choose your galaxy wisely, enjoy it as much or as little as you want, but keep your telescope pointed towards the other galaxies, and prepare to make a hyperjump to other places if needed.

3. Learn About Software History

This takes me to the next point: learn how your favorite technology came to be. Do you like C#? Do you know who created it? How did the .NET project came to be? Who was the lead architect? Which were the constraints of the project and why did the language turned out to be what it is now?

Apply the same recipe to any language or CPU architecture that you enjoy or love: Python, Ruby, Java, whatever the programming language; learn their origins, how they came up to be. The same for operating systems, networking technologies, hardware, anything. Go and learn how people came up with those ideas, and how long they took to grow and mature. Because good software takes ten years[76], you know.

> The mobile phone evolution @ValaAfshar[77] pic.twitter.com/ShP206GiYL[78]
>
> — JM Alvarez-Pallete (@jmalvpal) November 26, 2015[79]

The stories surrounding the genesis of our industry are fascinating, and will show you two things: first, that everything is a remix[80]. Second, that you could be the one remixing the next big thing. No, scratch that: you **are going to be** the creators of the next big thing.

And to help you get there, here is my (highly biased) selection of history books that I like and recommend:

- Dealers of Lightning[81] by Michael A. Hiltzik
- Revolution in the Valley[82] by Andy Hertzfeld

[76] http://www.joelonsoftware.com/articles/fog0000000017.html
[77] https://twitter.com/ValaAfshar?ref_src=twsrc%5Etfw
[78] https://t.co/ShP206GiYL
[79] https://twitter.com/jmalvpal/status/669781283732631553?ref_src=twsrc%5Etfw
[80] http://everythingisaremix.info
[81] http://www.amazon.com/Dealers-Lightning-Michael-A-Hiltzik-ebook/dp/B0029PBVCA
[82] http://www.amazon.com/Revolution-The-Valley-Paperback-Insanely/dp/1449316247

- The Cathedral and the Bazaar[83] by Eric S. Raymond
- The Success of Open Source[84] by Steven Weber
- The Old New Thing[85] by Raymond Chen
- The Mythical Man Month[86] by Frederick P. Brooks Jr.

You will also learn to value those things that stood the test of time: Lisp[87], TeX[88], Unix[89], bash[90], C[91], Cocoa[92], Emacs[93], Vim[94], Python[95], ARM[96], GNU make[97], man pages[98]. These are some examples of long-lasting useful things that are something to celebrate, cherish and learn from.

> I felt like saying this. pic.twitter.com/mHJ1rENoX1[99]
>
> — Hisham (@hisham_hm) December 13, 2015[100]

4. Keep on Learning

Learn. Anything will do. Wanna learn Fortran? Go for it. Find Erlang interesting? Excellent. Think COBOL might be the next big thing in your career? Fantastic. Need to know more about Functional Reactive Programming[101]? Be my guest. Design? Of course. UX? You must. Poetry? You should[102].

Many common concepts in Computer Science have been around for decades, which makes it worthwhile to learn old programming languages and frameworks; even “arcane” ones. First, it will make you appreciate the current state of the industry (or hate it, it depends,) and second, you will learn how to use the current tools more

[83] http://www.amazon.com/Cathedral-Bazaar-Musings-Accidental-Revolutionary/dp/0596001088
[84] http://www.amazon.com/Success-Open-Source-Steven-Weber/dp/0674018583
[85] http://www.amazon.com/Old-New-Thing-Development-Throughout/dp/0321440307/
[86] http://www.amazon.com/Mythical-Man-Month-Software-Engineering-Anniversary/dp/0201835959
[87] https://en.wikipedia.org/wiki/Lisp_%28programming_language%29
[88] https://en.wikipedia.org/wiki/TeX
[89] https://en.wikipedia.org/wiki/Unix
[90] https://www.gnu.org/software/bash/
[91] https://en.wikipedia.org/wiki/C_%28programming_language%29
[92] https://en.wikipedia.org/wiki/Cocoa_%28API%29
[93] https://www.gnu.org/software/emacs/
[94] https://en.wikipedia.org/wiki/Vim_%28text_editor%29
[95] https://en.wikipedia.org/wiki/Python_%28programming_language%29
[96] https://en.wikipedia.org/wiki/ARM_architecture
[97] https://www.gnu.org/software/make/
[98] http://manpages.bsd.lv/history.html
[99] https://t.co/mHJ1rENoX1
[100] https://twitter.com/hisham_hm/status/675845003709702144?ref_src=twsrc%5Etfw
[101] https://en.wikipedia.org/wiki/Functional_reactive_programming
[102] https://www.brainpickings.org/2014/02/17/joseph-brodsky-how-to-read-a-book/

effectively—if anything, because you will understand its legacy and origins.

Tip 1: **learn at least one new programming language every year.** I did not come up with this idea; The Pragmatic Programmer[103] book did. And it works.

One new programming language every year. Simple, huh? Go beyond the typical "Hello, World" stage, and build something useful with it. I usually build a simple calculator[104] with whatever new technology I learn. It helps me figure out the syntax, it makes me familiar with the APIs or the IDE, etc.

Tip 2: **read at least 6 books per year.** I have shown above a list of six must-read books; that should keep you busy for a year. Here goes the list for the second year:

- Peopleware[105] by Tom DeMarco and Tim Lister
- The Psychology of Software Programming[106] by Gerald M. Weinberg
- Facts and Fallacies of Software Engineering[107] by Robert L. Glass
- The Design of Everyday Things[108] by Don Norman
- Agile!: The Good, the Hype and the Ugly[109] by Bertrand Meyer
- Rework[110] by Jason Fried and David Heinemeier Hansson
- Geekonomics[111] by David Rice

(OK, those are seven books.)

Six books per year looks like a lot, but it only means one every 2 months. And most of the books I have mentioned in this presentation are not that long, and even better, they are outstandingly well written, they are fun and are full of insight.

Look at it this way: if you are now 20 years old, by the age of 30 you will have read over 60 books, and over 120 when you reach my age. And you will have played with at least 20 different programming languages. Think about it for a second.

Some of the twelve books I've selected for you have been written in the seventies, others in the eighties, some in the nineties and finally

[103] http://www.amazon.com/The-Pragmatic-Programmer-Journeyman-Master/dp/020161622X

[104] https://github.com/akosma/CodeaCalc

[105] http://www.amazon.com/Peopleware-Productive-Projects-Teams-Edition/dp/0321934113

[106] http://www.amazon.com/The-Psychology-Computer-Programming-Anniversary/dp/0932633420

[107] http://www.amazon.com/Facts-Fallacies-Software-Engineering-Robert/dp/0321117425

[108] http://www.amazon.com/The-Design-Everyday-Things-Expanded/dp/0465050654

[109] http://www.amazon.com/Agile-The-Good-Hype-Ugly/dp/3319051547

[110] http://www.amazon.com/Rework-Jason-Fried/dp/0307463745

[111] http://www.amazon.com/Geekonomics-Real-Insecure-Software-paperback/dp/0321735978

most of them are from the past decade. They represent the best writing I have come across in our industry.

But do not just read them; take notes. Bookmark. Write on the pages of the books. Then re-read them every so often. Borges[112] used to say that a bigger pleasure than reading a book is re-reading it. And also, please, buy those books you really like in paper format. Believe me. eBooks are overrated. Nothing beats the real thing.

Of course, please know that as you will grow old, the number of things that qualify as new and/or important will drop dramatically. Prepare for this. It is OK to weep silently when you realise this.

> If I could go back in time and tell the younger me exactly one and only one thing, it would be “learn UNIX”
>
> — Jeff Atwood (@codinghorror) February 4, 2016[113]

5. Teach

Once you have learnt, **teach**. This is very important.

This does not mean that you should setup a classroom and invite people to hear your ramblings (although it would be awesome if you did!) It might mean that you give meaningful answers to questions in Stack Overflow; that you write a book; that you publish a podcast about your favorite technology; that you keep a blog; that you write on Medium; that you go to another continent and set up programming schools using Raspberry Pis; or that you help a younger developer by becoming their mentor (do not do this before the age of 30, though.)

Teaching will make you more humble, because it will painfully show you how limited your knowledge is. **Teaching is the best way to learn**. Only by testing your knowledge against others are you going to learn properly. This will also make you more respectful regarding other developers and other technologies; every language, no matter how humble or arcane, has its place within the Tao of Programming[114], and only through teaching will you be able to feel it.

And through teaching you can really, really make a difference in this world. Back in 2012 I received a mail from a person who had attended one of my trainings. She used to work as an Adobe Flash developer. Remember ActionScript and all that? Well, unsurprisingly[115] after 12 years of working as a freelance Flash developer she suddenly found herself unemployed. Alone. With a baby to feed. She told me in her message that she had attended my training, that she had enjoyed it

[112] https://en.wikipedia.org/wiki/Jorge_Luis_Borges

[113] https://twitter.com/codinghorror/status/695072624322293760?ref_src=twsrc%5Etfw

[114] http://canonical.org/~kragen/tao-of-programming.html

[115] https://web.archive.org/web/20200430094807/https://www.apple.com/hotnews/thoughts-on-flash/

and also learnt something useful, and that after that she had found a job as a mobile web developer. She wrote to me to say thank you.

I cannot claim that I changed the world, but I might have nudged it a little bit, into something (hopefully) better. This thought has made every lesson I gave since then much more worthwhile and meaningful.

6. Workplaces Suck

> Every day, with every action and choice, you're either a teacher and an inspiration, or a lesson and a reminder.
>
> — Cat Swart (@Jexx) May 25, 2015[116]

Do not expect software corporations to offer any kind of career path. They might do this in the US, but I have never seen any of that in Europe. This means that you are solely responsible for the success of your career. Nobody will tell you "oh, well, next year you can grow to be team leader, then manager, then CTO…"

Not. At. All. Quite the opposite, actually: you were, are and will be a software developer, that is, a relatively expensive factory worker, whose tasks your managers would be happy to offshore no matter what they tell you.

Do not take a job just for the money. Software companies have become sweatshops[117] where you are supposed to justify your absurdly high salary with insane hours and unreasonable expectations. And, at least in the case of Switzerland, there is no worker union to help you if things go bad. Actually there are worker unions in Switzerland, but they do not really care about situations that will not land them some kind of media exposure.

Even worse; in most workplaces you will be harassed, particularly if you are a woman, a member of the LGBT community or from a non-caucasian ethnic group. I have seen developers threatened to have their work visas not renewed if they did not work faster. I have witnessed harassment of women and gay colleagues.

Some parts of our industry are downright disgusting, and you do not need to be in Silicon Valley to live it. You do not need Medium to read it. You could experience that right here in Switzerland. Many banks have atrocious workplaces. Financial institutions want you to vomit code 15 hours a day, even if the Swiss working laws explicitly forbid such treatments. Pharmaceutical companies want you to write code to cheat test results and to help them bypass regulations. Startups want your skin, working for 18 hours for no compensation, telling you bullshit like "because we give you stock options" or "because we are all team players."

[116] https://twitter.com/Jexx/status/602943242024390656?ref_src=twsrc%5Etfw
[117] https://twitter.com/carlfish/status/721738272070762496

It does not matter that you are Zach Holman and that you can claim in your CV that you literally wrote Github from scratch: you will be fired[118] for the pettiest of reasons.

It does not matter that the app brings more than half of your employer traffic and revenues; the API team will treat you and your ideas with contempt and sloppiness.

I have been asked to work for free by very well known people in the industry, some of them even featured in Wikipedia, and it is simply appalling. I will not give out their names, but I will prevent any junior from getting close to them, because people working without ethics **do not deserve anyone's brain.**

Whenever an HR manager tells you "you must do this (whatever wrong thing in your frame of reference) because we pay you a salary," remember to answer the following: "you pay me a salary, but I give you my brain in exchange, and I refuse to comply with this order."

And to top it all, they will put you in an open space, and for some reason they will be proud about it. **Open spaces are a cancer.** They are without a doubt the worst possible office layout ever invented, and the least appropriate for software development—or any type of brain work for that matter.

Remember this: the fact that you understand something does not imply that you have to agree to it.

Disobey authority. Say "fuck you, I won't do what you tell me"[119] and change jobs. There are fantastic workplaces out there; not a lot, but they exist. I have been lucky enough to work in some of them. Do not let a bad job kill your enthusiasm. It is not worth it. Disobey and move on.

Or, better yet, become independent.

> Myth: Open offices result in massive collaboration.
>
> Reality: 2 people loudly collaborate; 30 must wear headphones to get any work done.
>
> — Jochen Wolters (@jochenWolters) April 7, 2016[120]

7. Know Your Worth

You have probably heard about the "10x Software Engineer" myth, right? Well here is the thing: it is not a myth, but it does not work they way you think it works.

It works, however, from the point of view of the employer: a "10x Software Engineer" generates worth 10 times whatever the employer

[118] https://zachholman.com/talk/firing-people

[119] https://www.youtube.com/watch?v=bWXazVhlyxQ

[120] https://twitter.com/jochenWolters/status/718175220637392897?ref_src=twsrc%5Etfw

pays. That means that you she or he gets 100 KCHF per year, but she or he are actually creating a value worth over a million francs. And of course, they get the bonuses at the end of the fiscal year, because, you know, capitalism. Know your worth. Read Karl Marx[121] and Thomas Piketty[122]. Enough said.

Keep moving; be like the shark that keeps on swimming, because your skills are extremely valuable. Speak out your salary, say it loud, blog about it, so that your peers know how much their work is worth. Companies want you to shut up about that, so that women are paid 70% of what men are paid. So speak up! Blog about it! Tweet it! I am making 135 KCHF per year. That was my current salary. How about you? And you? The more we speak out, the less inequality there will be. Any person doing my job with my experience should get the same money, regardless of race, sex, age or preferred football team. End of the story. But it is not like that. It is not.

> A customer walks into a bar. He asks for a beer made out of wine. The project manager agrees. Both question the bartender's competence.
>
> — Daniel Méndez (@mendezfe) March 22, 2015[123]

8. Send The Elevator Down

If you are a white male remember all the privilege you have enjoyed since birth just because you were born that way. It is your responsibility to change the industry and its bias towards more inclusion.

It is your **duty** to send the elevator down.

Take conscious decisions in your life. Be aware of your actions and their effect. Do not blush or be embarrased for changing your opinions. Say "I'm sorry" when required. Listen. Do not be a hotshot. Have integrity and self-respect.

Do not critisize or make fun of the technology choices of your peers; for other people will have their own reasons to choose them, and they must be respected. Be prepared to change your mind at any time through learning. One day you might like Windows. One day you might like Android. I am actually liking some parts of Android lately. And that is OK.

> Desirable developer skills:
>
> 1 Ability to ignore new tools and technologies 2 Taste for simplicity 3 Good code deletion skills 4 Humility

[121] http://www.amazon.com/Capital-Critique-Political-Economy-Classics/dp/0140445684

[122] http://www.amazon.com/Capital-Twenty-First-Century-Thomas-Piketty/dp/067443000X

[123] https://twitter.com/mendezfe/status/579559198247260160?ref_src=twsrc%5Etfw

Figure 3: Nothing is as simple as it seems at first, or as hopeless as it seems in the middle, or as finished as it seems in the end

> — David Winterbottom (@codeinthehole) December 3, 2014[124]

9. LLVM

Everybody is raving about Swift, but in reality what I pay more attention to these days is LLVM itself.

I think LLVM is the **most** important software project today, as measured in its long-term impact. Objective-C blocks, Rust & Swift (the two most loved strongly typed and compiled programming languages in the 2016 StackOverflow developer survey[125],) Dropbox Pyston[126], the Clang Static Analyser, ARC, Google Souper[127], Emscripten[128], LLVMSharp[129], Microsoft LLILC[130], Rubymotion[131], cheerp[132], watchOS apps, the Android NDK[133], Metal[134], all of these things were born out or powered by LLVM. There are compilers using LLVM as a backend for pretty much all the most important languages of today. The .NET CLR will eventually interoperate[135] with it, and Mono already[136] uses it. Facebook has tried to integrate LLVM with HHVM[137], and WebKit recently switched from LLVM to the new B3 JIT JavaScript compiler[138].

LLVM is cross-platform[139], cross-CPU-architecture, cross-language, cross-compiler, cross-eyed-tested, free as in gratis and free as a bird.

Learn all you can about LLVM. This is the galaxy where true innovation is happening now. This is the foundation for the next 20 years.

> @owensd[140] Java is 20 years old, C# is 15 – I think it is better to see Swift as a response to that sort of managed language (the next step?)
>
> — Chris Lattner (@clattner_llvm) February 18, 2015[141]

---

[124] https://twitter.com/codeinthehole/status/540117725604216832?ref_src=twsrc%5Etfw
[125] http://stackoverflow.com/research/developer-survey-2016
[126] https://github.com/dropbox/pyston
[127] https://github.com/google/souper
[128] https://kripken.github.io/emscripten-site/
[129] http://llvmsharp.org/
[130] https://github.com/dotnet/llilc
[131] http://www.rubymotion.com/tour/features/
[132] http://leaningtech.com/cheerp/
[133] https://developer.android.com/ndk
[134] https://developer.apple.com/metal/
[135] https://github.com/dotnet/llilc
[136] https://www.mono-project.com/docs/advanced/mono-llvm/
[137] http://hhvm.com/blog/10205/llvm-code-generation-in-hhvm
[138] https://webkit.org/blog/5852/introducing-the-b3-jit-compiler/
[139] http://llvm.org/docs/doxygen/html/classllvm_1_1Triple.html
[140] https://twitter.com/owensd?ref_src=twsrc%5Etfw
[141] https://twitter.com/clattner_llvm/status/568128101625933824?ref_src=twsrc%5Etfw

10. Follow Your Gut

I had the gut feeling .NET was going to be big when I watched its introduction in June 2000[142]. I had the gut feeling the iPhone was going to be big when I watched its introduction in 2007[143].

In both cases people laughed at my face, literally. In both cases I followed my gut feeling and I guess things worked out well.

Follow your gut. You might be lucky, too.

> Follow your heart Follw yur heart Fllw yr hart Fw y art Fart
>
> — Daniel Kibblesmith (@kibblesmith) November 17, 2014[144]

11. APIs Are King

Great APIs enable great apps. If the API sucks, the app will suck, too, no matter how beautiful the design.

Remember that **chunky is better than chatty**, and that clients should be dumb; push as much logic as you can down to the API.

Do not invent your own security protocols.

Learn a couple of server-side technologies, and make sure Node is one of those.

Leave REST aside and embrace Socket.io, ZeroMQ, RabbitMQ, Erlang, XMPP; explore realtime as the next step in app development. Realtime is not only for chat apps. Remove polling from the equation forever.

Oh, and start building bots[145] around those APIs. Just saying.

12. Fight Complexity

Simpler is better. Always. Remember the KISS principle. And I do not mean only at the UI level, but all the way until the deepest layers of your code.

Refactoring, unit tests, code reviews, pull requests, all of these tools are at your disposal to make sure that the code you ship is the simplest possible architecture that works. This is how you build resilient systems for the long term.

> Follow your heart Follw yur heart Fllw yr hart Fw y art Fart

---

[142] http://www.zulenet.com/see/BillGatesNET.html

[143] https://www.youtube.com/watch?v=9hUIxyE2Ns8

[144] https://twitter.com/kibblesmith/status/534421408575922176?ref_src=twsrc%5Etfw

[145] http://www.economist.com/news/business-and-finance/21696477-market-apps-maturing-now-one-text-based-services-or-chatbots-looks-poised

Figure 4: The impossible roundabout

— Daniel Kibblesmith ☃ (@kibblesmith) November 17, 2014[146]

Conclusion

The most important thing to remember is that your age does not matter.

> One of my sons said to me, “Impossible, Dad. Mathematicians do all their best work by the time they’re 40. And you’re over 80. It’s impossible for you to have a good idea now.”
>
> If you’re still awake and alert mentally when you’re over 80, you’ve got the advantage that you’ve lived a long time and you’ve seen many things, and you get perspective. I’m 86 now, and it’s in the last few years that I’ve had these ideas. New ideas come along and you pick up bits here and there, and the time is ripe now, whereas it might not have been ripe five or 10 years ago.
>
> Michael Atiyah, Fields Medal and Abel Prize winner Mathematician, quoted in a Wired article.

[146] https://twitter.com/kibblesmith/status/534421408575922176?ref_src=twsrc%5Etfw

As long as your heart tells you to keep on coding and building new things, you will be young, forever.

In 2035, exactly 19 years from now, somebody will give a talk at a software conference similar to this one, starting like this:

“Hi, I am 42 years old, and this is my story.”

Hopefully one of you will be giving that presentation; otherwise, it will be an AI bot. You will provide some anecdotical facts about 2016, for example that it was the year when David Bowie[147], Umberto Eco[148], Gato Barbieri[149] and Johan Cruyff[150] passed away, or when SQL Server was made available in Linux[151], or when Google AlphaGo[152] beat a Go champion, or when the Panama Papers[153] and the Turkish Citizenship Database[154] were leaked the same day, or when Google considered using Swift for Android for the first time[155], or as the last year in which people enjoyed this useless thing called privacy.

We will be three years away from the Year 2038 Problem[156] and people will be really nervous about it.

Of course I do not know what will happen 19 years from now, but I can tell you three things that will happen for sure:

1. Somebody will ask a question in Stack Overflow about how to filter email addresses using regular expressions.
2. Somebody will release a new JavaScript framework.
3. Somebody will build something cool on top of LLVM.

And maybe you will remember this talk with a smile.

Thank you so much for your attention.

[147] https://en.wikipedia.org/wiki/David_Bowie
[148] https://en.wikipedia.org/wiki/Umberto_Eco
[149] https://en.m.wikipedia.org/wiki/Gato_Barbieri
[150] https://en.wikipedia.org/wiki/Johan_Cruyff
[151] https://blogs.microsoft.com/blog/2016/03/07/announcing-sql-server-on-linux/
[152] https://en.wikipedia.org/wiki/AlphaGo
[153] http://panamapapers.sueddeutsche.de/en/
[154] http://adam.curry.com/art/1459781684_9D9NHCND.html
[155] http://www.macrumors.com/2016/04/07/google-possibly-adopting-swift-for-android/
[156] https://en.wikipedia.org/wiki/Year_2038_problem

Developers, Learn To Say No

Adrian Kosmaczewski

2016-05-12

All change starts with a “No.”

This morning I started my day with a dozen crashes all over the various devices that our technological life has sprinkled around us. First an app crashed on my iPad, then a web page stopped responding in Chrome, then Safari also had its hiccups, finally an iPhone app stopped responding. A couple of reboots followed.

And I have not opened Xcode yet.

Suffice to say, I had enough, and I hyperventilated my frustration on Twitter, begging for the care and support of my dear followers. Not a good start for my day.

> we’ve reached the point in mankind history where every. single. piece. of. software. is. broken.
>
> not even a webpage without bugs, crashes…
>
> — akosma (@akosma) May 12, 2016[1]

“Software is eating the world[2],” they say. “Now every company is a software company[3],” they say. And I wish I could build a time machine and seek refuge somewhere in the middle of the Pleistocene[4] era. Alas, we are not there yet, and anyway, given the current state of software, I would end up in the middle of the Trinity test[5] instead.

The problem is not technology, however. It does not matter which programming language you use. It does not matter which operating system your code runs on. The problem lies in us, software developers all over the planet, because **we say “Yes” all too often.**

I have a cure for that.

We have to start saying “No,” louder and more often than ever.

[1] https://twitter.com/akosma/status/730653196474044416?ref_src=twsrc%5Etfw

[2] http://www.forbes.com/sites/stevedenning/2014/04/11/why-software-is-eating-the-world/#3a7fed7d3b24

[3] http://www.forbes.com/sites/techonomy/2011/11/30/now-every-company-is-a-software-company/#72cd9d4c1100

[4] https://en.wikipedia.org/wiki/Pleistocene

[5] https://en.wikipedia.org/wiki/Trinity_%28nuclear_test%29

A Call To Action

We should start right now.

See that manager approaching your cubicle? He will ask you in a few minutes for a “rough estimation” of a new feature.

Say “No.”

See that meeting room? Project managers and product owners are asking you whether a feature will soon be ready or not.

Say “No.”

There is a manager, right now, somewhere, trimming next year’s budget of the QA team, because whatever.

Say “No.”

The Reasons We Say “Yes”

We live in a world where, apparently, everything has to be positive. Ping pong tables. Free lunches. Money. Segways. Tesla. “Aim for the stars, you might end up on the Moon.” “Yes we can!” “Just do it!”

We are expected to have a “can-do” attitude, and to always bring positive ideas to the table, to contribute to a “positive team spirit,” that “we can,” and that, therefore, “we must.”

Now, please, read the following article carefully, and come back as soon as you have finished: “We Have Become Exhausted Slaves in a Culture of Positivity.[6]” I think the title says it all.

The common wisdom states that young developers, freshly graduated and early in their careers are more prone to say “Yes,” but this is a misguided view, as I have been myself found guilty of saying “Yes” too quickly recently, and I have heard it very often from other senior developers, too. Saying “Yes” can happen, and it is sometimes unfortunate. It can have a terrible effect in project schedules, in project costs, and in the final quality of products—or rather, their lack thereof.

The Fear Of Saying “No”

The net result of this situation is that we fear saying no. We fear rejection. We fear being left out. I know about that: I have been rejected, I have been left out because of my tendency to be rather pessimistic in my estimations and my outlook.

Having experience, having “been there, done that,” being the “senior developer” of a team should and can include a safe amount of conservative views. What is not okay is stubbornness and failing to learn from evidence; being senior does not mean being always right. But

[6] http://www.scilogs.com/next_regeneration/we-have-become-exhausted-slaves-in-a-culture-of-positivity/

being conservative is a totally acceptable characteristic. I would even go as far and say, it is a badly needed one.

Of course, saying no will set you aside. You are going to be left out from important meetings. You might even be fired because of your "attitude" or your "lack of integration with the team spirit" or whatever the buzzword of the day in Human Resources circles.

And you know what? You should leave. Go away. As I mentioned in my previous article[7], the software industry has an incredible demand for software developers, and an incredibly short supply thereof. So leave. You do not need to stay in a place where your views are systematically rejected "because positivity." **Life is short, and you must feel confident to speak out your mind regarding project issues.**

The Virtuosity Of Saying "No"

There is, however, a happy ending, one that I have witnessed in the past, albeit not very often: when you say "No," occasionally, somebody will look at you in the eyes, and with a puzzled look will ask you **"Why?"**

Aha! I love that moment. Hear the angels singing "Hallelujah[8]" as they come down from the skies? You have finally struck the chord. Something important is about to happen. Breathe and smile. This is a great moment in your career.

At that moment, look at the person in the eyes, and I hope you will realize that the question was fair, frank and genuine. If that is the case, breathe deeply once again, and provide all the rationale you can about your point of view.

Explain that the feature will take longer than expected because the QA budget has been reduced. Or because the continuous integration server has to be upgraded. Or because you need more VMs in Amazon or Azure. Or because there are version compatibility problems that could appear. Or because you need more developers, et cætera.

You will know if the person in front of you wants to learn, or if it is just a hypocrite question. You will know. If the question does not fall in the latter case, it is your **obligation** to explain your point of view. Give all the information you have. Be generous. Draw a couple of diagrams. Teach. Be the senior developer that you are meant to be. This person then should explain to you the business background, the reasons behind the situation, the rationale for some decisions, and all those details that you might not be aware of.

See what is happening now? A conversation, a real dialogue between business and technology. And now you know why it happened: **because you said "No."**

[7] /blog/being-a-developer-after-40/

[8] https://www.youtube.com/watch?v=UCl4ZHrJSVo

So, go on, say it. Loud.
Do not be afraid of saying “No.”
And then teach, learn, rinse and repeat.

Una Habitacion Con Vista Al Mar

Adrian Kosmaczewski

2016-06-13

Intentando algo nuevo, he aqui un screenplay para una película. Una que probablemente nunca se haga, pero no importa.

Se trata de una adaptacion (no muy fiel, para ser sinceros) de un cuento de hernun[1] en forma de libreto para cine. Se puede leer aqui mismo[2] (tambien en pdf[3] y en fountain[4]).

[1] http://hernun.com.ar/

[2] /books/Una_Habitaci%C3%B3n_Con_Vista_Al_Mar/Una_Habitaci%C3%B3n_Con_Vista_Al_Mar.html

[3] /books/Una_Habitaci%C3%B3n_Con_Vista_Al_Mar/Una_Habitaci%C3%B3n_Con_Vista_Al_Mar.pdf

[4] /books/Una_Habitaci%C3%B3n_Con_Vista_Al_Mar/Una_Habitaci%C3%B3n_Con_Vista_Al_Mar.fountain

You, Founder, and your Secret Agenda

Adrian Kosmaczewski

2016-06-16

So, dear Founder of a much hyped techno bubble, you want my skills in your company.

Of course. I have been writing code probably for longer than you have been alive. And you believe you have the power to change the world through your… whatever… gizmo that you have just invented. Well, that you have somehow came up with and have hired a team to make it real, and of course it is only your name on the front page of the website.

Let me tell you something, dear nothing: No, I will not work with you. The reasons are various, deep and interesting, and I assume that if you are still reading this article it is precisely because you are interested in learning which are these reasons.

First of all, you are a hypocryte. You do not want to change the world, but actually just to become rich. In the best case, that is. Because in the worst, your hubris tells you that you deserve to dominate the world in some extent. You want to singlehandedly ripe all benefits in some industry, sector, whatever. You want to be filthy rich. You cannot have enough money. But you do not have the technical knowledge to do that, so you temporarily pretend to be my friend so that I can bring your ideas to life.

Hubris. Yes, you might want to check the meaning of that word on a dictionary.

You know why so many industries are disrupted? Not because of good ideas, no. But because good men have let people like you take the lead and dictate the course of history.

As software engineer, I take the conscious decision to stop helping you. I am tired of your hypocrisy. Just do not pretend to be my friend, say openly what your plans are. I do not want your fusball table on the lobby of the hipster neighborhood offices. I do not want your pizza evenings. I do not want your babble about your vision of the world, or any other shit.

Keep that to yourself. And believe me, not only will I avoid working with you, I will do my best to make any decent software engineer to work for you as well.

The time for a conscience change has arrived, and we as a profession we need to stop enabling idiots like you to change the world for the worse.

No, Uber is not making the world a better place. Neither is Nest, Google, Apple or any other big corporation. The only thing Silicon Valley wants is money and, of course, its bastard sibling: power.

Do not ask for a reconsideration or a change of mind. I will fight your idiocy and your hypocrisy with as much energy as I can, waking up the young generations you are trying to convince about some greater good.

There is no greater good. There is only your hubris, hidden beneath a veil of hypocrisy.

You have been warned.

Eight Steps To Build A Better Swiss Software Industry

Adrian Kosmaczewski

2016-06-22

The Swiss are the best at many, many different domains. Which is a rather surprising and seriously fantastic feat for such a small country with merely eight million people, divided in four quite distinct linguistic groups.

For example, in no particular order, let us mention those incredible artisan watchmakers[1], from La Chaux-de-Fonds[2] to Schaffhausen[3], creating incredibly complex small machines almost entirely by hand, passing knowledge and tradition to newer generations for almost 300 years, and able to overcame serious situations like the "Quartz Crisis"[4] in the seventies and eighties.

Or those amazing civil engineers, able to design and build the longest and deepest tunnels[5] ever dug in the planet, able to build one of the highest dams[6] in the world in the middle of the Alps, or able to lay down train tracks, aerial tramways or bridges in the most unusual and breathtaking[7] locations on Earth.

Or these outstanding tennis players, constantly in the higher rankings for the past 25 years, like Marc Rosset[8], Martina Hingis[9], Roger Federer[10], Stan Wawrinka[11] or Timea Bacsinszky[12].

Or those fantastic adventurers, explorers and entrepreneurs, like Jean Louis Burckhardt[13], first European to find the city of Petra. Sarah Marquis[14], who walked twenty thousand kilometers from Siberia to Aus-

[1] https://en.wikipedia.org/wiki/Swiss_made
[2] https://en.wikipedia.org/wiki/La_Chaux-de-Fonds
[3] https://en.wikipedia.org/wiki/Canton_of_Schaffhausen
[4] https://en.wikipedia.org/wiki/Quartz_crisis
[5] https://en.wikipedia.org/wiki/Gotthard_Base_Tunnel
[6] https://en.wikipedia.org/wiki/Grande_Dixence_Dam
[7] https://en.wikipedia.org/wiki/Landwasser_Viaduct
[8] https://en.wikipedia.org/wiki/Marc_Rosset
[9] https://en.wikipedia.org/wiki/Martina_Hingis
[10] https://en.wikipedia.org/wiki/Roger_Federer
[11] https://en.wikipedia.org/wiki/Stan_Wawrinka
[12] https://en.wikipedia.org/wiki/Timea_Bacsinszky
[13] https://en.wikipedia.org/wiki/Johann_Ludwig_Burckhardt
[14] https://en.wikipedia.org/wiki/Sarah_Marquis

tralia. Claude Nicollier[15], the astronaut who led the Space Shuttle team to cure Hubble from myopia[16]. Or the Piccard family, who in three generations went from the deepest ends of the ocean[17], to the highest levels of the atmosphere[18], to travel around the world in an electric airplane[19]. Or Louis Chevrolet[20], pioneer of the automotive industry and founder of the car brand bearing his name.

And let us not forget about renowned artists like Sophie Taeuber-Arp[21], Alberto Giacometti[22], Mario Botta[23], Le Corbusier[24], Jean Tinguely, or Patrick Chappatte[25], featuring their work in the most important museums of the world and considered references in their respective crafts.

And I could continue on and on and on. Women and men who have shaped the world, crossed borders, enlightened our imaginations and showed the world that the sky is not even a limit, all coming from a small landlocked country, stubbornly independent and filled with cheese, watches, Heidis and other clichés.

What About Software?

Apart from some glorious exceptions, among which I would name the venerable Niklaus Wirth[26], creator of Pascal among other languages; Martin Odersky[27], the creator of Scala, from German origin but teacher at the EPFL[28] in Lausanne; Alexandre Juillard[29]; René Sommer[30]; or companies like Kudelski[31] or Logitech[32], the Swiss landscape of software engineering is definitely not what one would expect at first, particularly when compared to other branches of engineering.

How many Swiss-made software products make the headlines at Techcrunch? How many Swiss startups are bought every year by big companies in Silicon Valley with valuation in billions? How many Swiss software developers have cashed their stock options

[15] https://en.wikipedia.org/wiki/Claude_Nicollier
[16] https://en.wikipedia.org/wiki/STS-61
[17] https://en.wikipedia.org/wiki/Jacques_Piccard
[18] https://en.wikipedia.org/wiki/Auguste_Piccard
[19] https://en.wikipedia.org/wiki/Bertrand_Piccard
[20] https://en.wikipedia.org/wiki/Louis_Chevrolet
[21] https://en.wikipedia.org/wiki/Sophie_Taeuber-Arp
[22] https://en.wikipedia.org/wiki/Alberto_Giacometti
[23] https://en.wikipedia.org/wiki/Mario_Botta
[24] https://en.wikipedia.org/wiki/Le_Corbusier
[25] https://en.wikipedia.org/wiki/Patrick_Chappatte
[26] https://en.wikipedia.org/wiki/Niklaus_Wirth
[27] https://en.wikipedia.org/wiki/Martin_Odersky
[28] https://www.epfl.ch/index.en.html
[29] https://en.wikipedia.org/wiki/Alexandre_Julliard
[30] https://en.wikipedia.org/wiki/Ren%C3%A9_Sommer
[31] https://www.nagra.com/
[32] http://www.logitech.com/

so far, à part those who had the luck of working at Logitech in the eighties? How many Swiss app makers have won an Apple Design Award[33] (apart from Pixmeo[34] for their product OsiriX[35] in 2005, or Curvus Pro X, runner-up in 2004)?

The answer to all of those questions is a grim mix between ”not a lot” and ”none at all.” The so often called “Swiss Silicon Valley” is yet to be found.

This lack of brilliance is mostly apparent in the overall quality, usability and usefulness (or, rather, lack thereof) of Swiss software products. These are most usually late to market, feature clunky and awkward user interfaces, are usually more expensive than their counterparts and are designed with the least possible common sense. Thankfully most of these products are rarely exported, limiting their possibility for mental disruption to the borders of this country.

This dire situation has been identified long ago; to solve this dilemma, many different ”startup incubators” exist in the country, both in the German- and French-speaking regions, from federal to cantonal level, to encourage the creation of new companies in buzzword-compliant “hot” industries such as “Wearables,” “Fintech,” “IoT” or other similar ones.

But, in spite of the billions of Swiss Francs injected into these hubs, one would be hard-pressed to watch the spectacular expected outcome, the one your favorite financial magazine regularly fills its pages with.

Why is that? What makes Switzerland a relatively poor choice for technology startups, particularly when compared to other European hubs like Berlin, Amsterdam or London? Why is the Swiss software industry not following the steps of other branches of engineering and being showcased as world-class examples of design, usefulness and universality? Why are so many Swiss star developers leaving the country to shine abroad?

The Reasons

I have identified ten basic reasons for this situation, and they have all to do with Swiss culture:

1. Hierarchies
2. Design-by-committee
3. Cult Of The Monoculture
4. Lack Of Technical Knowledge Of Project Managers And Founders
5. Lack Of Digital Design Education
6. Excessive Quest For Perfectionism
7. Lack Of Proper Risk Management

[33] https://en.wikipedia.org/wiki/Apple_Design_Awards
[34] http://pixmeo.pixmeo.com/
[35] http://www.osirix-viewer.com/

8. Excessive Processes Considered Harmful

As Jack the Ripper would have said, let's go by parts.

1. Hierarchies

Hierarchy is, by far, the primary social organization of Switzerland.

Everything in Swiss culture has to fit in some kind of pyramidal structure, and this has been the case for centuries. The most important example of this mindset is none other than the Swiss Army[36], one of the largest militia armies in the planet, rigidly structured and where the life and work of every single soldier is clearly defined, managed and observed.

Let us not forget that, still twenty or even fifteen years ago, one precondition to be promoted as a manager in any Swiss company was to be an officer in the Swiss army. Things have changed (thankfully!) but in many small firms, the idea of an all-knowing superior mind on top of the hierarchy still exists, and still drives people.

In a hierarchy, there is a clear tradeoff: **control versus dialog.** In a hierarchy, the workers at the bottom of the food chain are not expected to be listened to. And if by some level of corporate hypocrisy they are made to believe otherwise, the top management can afford to avoid listening at any moment.

Decision makers are on top. They are always right, even they are not. They are the captains of the ship, until it sinks.

The top management layer of many Swiss companies is utterly incompetent in technical matters (more on this later) and is completely and absolutely incapable of taking technical decisions; yet, that is what they do. Most "suits" have beautiful MBA degrees, awesome PMI certifications, receive incredibly high salaries and "status perks," all while taking horrendous decisions, later holding absolutely no accountability whatsoever for their mishaps.

The net result is that there is a huge level of turnover among Swiss software developers, simply because top management will never listen to them, because there is no dialog, only command-and-control structures, and so they leave. And turnover is expensive. Very, very expensive.

Hierarchies are the first and most important problem in Swiss software development companies, because **software development is a social process.** It requires dialog, feedback, from all levels of the organization, and without that feedback, there cannot be quality software at the end of the day.

This situation must change, and I hope that the few "suits" reading this text will realize that the golden cage where they live is actually

[36] https://en.m.wikipedia.org/wiki/Military_of_Switzerland

keeping Switzerland back from becoming a world leader in software engineering.

The solution is simple: stand down from your thrones, trust your engineers and learn from them. And if this is the only factor that you care about, please know that you could be making far more money than you already do if you followed this simple mantra in your organization.

Swiss software makers must drop hierarchies.

2. Design-by-Committee

Switzerland is an incredibly democratic country, featuring elements like the right to propose popular initiatives[37], regular referendums and a decentralized power system (organized in quite a tight hierarchy[38] from cantons to districts to municipalities) which is unique in the world.

However, when it comes to product management, democracy is a hindrance. And when it is coupled to hierarchies, you get the second biggest problem of the software industry in Switzerland: design by committee.

Product management is a complex thing, and for accountability purposes (remember hierarchies?) Swiss companies tend to organize decision making in product teams around groups of people, literally voting (not a joke) for end user features, or for the color and fonts used in the UI.

This situation is actually holding back Swiss designers from being more recognized abroad (more on that later) but it also generates at the end absolutely horrendous products, simply unusable and lacking the most basic common sense.

Product teams should have two things: **a strong opinion and accountability.** There should be one, maybe two people at most taking decisions, and these people should have both design and engineering skills. For products designed by non-technical designers will be technically impossible to create, and products created by design illiterate engineers will look and feel horrible.

Great products are born out of a dialog between design and engineering (remember what I said about hierarchies trading dialog for control in the previous section) together with opinion and accountability. Right now this dialog is not happening in this country.

Swiss software makers must drop hierarchies, to enable a dialog between design and engineering, using a more opinionated and less democratic approach to product management.

[37] https://en.wikipedia.org/wiki/Popular_initiative_(Switzerland)
[38] https://en.wikipedia.org/wiki/Subdivisions_of_Switzerland

3. Cult Of The Monoculture

For all of its incredible democracy, Switzerland was one of the last countries in the world to allow women to vote[39] in Federal elections, as late as 1971. For all of its incredible diversity, this is a country where a right-wing party owns a newspaper, the infamous German-speaking Weltwoche, which regularly mocks the French side of the country (itself featuring higher GDP growth rates than the German cantons for the past decade) as an underdeveloped poor peninsula.

The Swiss engineering culture is slowly changing and moving away from stereotypes, but still shows a strong bias towards teams formed exclusively by white males. Even worse, most of them usually come from the same school or the same canton and speak the same dialect. This is happening in a country where 50% of the people are women, where 40% of the inhabitants come from abroad, where there are four official national languages, where 35% of the population at least does not speak German properly, and where there is a myriad of technical schools in every corner of the country.

Even worse, the hierarchies in place actively discourage or blatantly dismiss input from women, from people from different nationalities (even from different cantons…,) from engineers from different schools or from different age groups. And this leads to an unspoken series of despicable situations, among which sexual and moral harassment, discrimination based on gender or age or nationality, abhorrent gender salary gaps, and a myriad of lost opportunities for our economy. All of this is increasing turnover rates, and in the worst cases leads to burnout, illnesses and sadly even to suicides.

Why are not we speaking about these issues? Why are not we changing these attitudes? What is it that we Swiss are so proud about this outrageous engineering culture? We should be ashamed of ourselves.

This situation is easily visible in the all-white-males “about us” pages in most websites – where sometimes women appear as receptionists or HR managers.

It is also blatantly visible in large number of websites and software packages produced in Switzerland in **only one national language.** If the product happens to be developed in Geneva, forget about German or Italian; if the product happens to come from Zurich, forget about French or Italian. Actually, you can easily forget about Italian and Rumantsch[40], the fourth national language, spoken by around forty thousand people, but completely out of the horizon for virtually every single company or product online.

And for a country that has such a large number of expats with an incredible purchasing power at their disposal, the number of commercial websites in English is abysmally low. Talk about lost opportunities.

[39] https://en.wikipedia.org/wiki/Women%27s_suffrage_in_Switzerland

[40] https://en.wikipedia.org/wiki/Romansh_language

Switzerland is indeed a small market, but if your product is only available in German or French, you are making your target market even smaller.

And let us not even talk about software accessibility issues, which are hardly ever taken into consideration by our industry.

We need to break the Swiss monoculture, in order to create an inclusive polyculture. We need more women, more foreigners, more languages, more younger and older people working together. This diversity is one of our greatest chances for success, and it only exists as such in a few other countries in the world. This diversity will be the key to make the best possible software products in the planet, and will truly place Switzerland at the forefront of the software industry.

Swiss software makers must drop hierarchies, to enable a dialog between design and engineering, using a more opinionated and less democratic approach to product management; including people from all origins, sexual orientations, ages, cultures and languages.

4. Lack Of Technical Knowledge Of Project Managers And Founders

Software is a complex thing. Some say it is probably the most complex of all the creations ever done by mankind. Software allows us to land a small probe on a comet traveling at incredible speed at the far end of the solar system. Software allows us to talk, to communicate, to laugh, to share, to create, in as many ways as possible.

To achieve all of that complexity, lots of people have to coordinate efforts, during weeks or months, sometimes taking steps back to ensure that the much feared “technical debt” will not end up eating the whole of the budget.

But for some reason, most directors, project managers, product owners, product managers in this country actively believe that they can manage engineers from the top of their comfortable hierarchies without any technical knowledge whatsoever. Heck, they even brag about this fact; and cluelessly enough, they even do it in front of their own engineers.

This is a situation that cannot continue. Particularly in a country with hierarchies replacing all dialog with rigid control procedures. Particularly in an industry that requires complex technical decisions at every single small step. This is a recipe for disaster and this is the daily situation of most Swiss software engineers.

The solution for this problem is twofold.

On one side, MBA programs should all include by now a required entry in their curriculum, to teach their students how to manage software projects. These kind of projects are unlike any other, and feature

team dynamics that can be unsettling and unfamiliar. Yet, stubbornly enough, this is not happening.

We need managers explaining the current business problems to the developers; they can understand those problems, and they will thank you for establishing an open channel between different teams in the same company. Do not hide business problems to your engineering team; be transparent to them.

On the other side, we need developers to start teaching their project and product managers and even their CEOs, that it is tantamount for the success of their teams to have them understand the challenges that are underpinning every technology choice.

We need developers saying no[41], once and again, to managers in the top levels of the hierarchies that are oblivious to the technicalities of the projects that will pay for their bonuses at the end of the year.

Swiss software makers must drop hierarchies, to enable a dialog between design and engineering, that is between managers and developers; using a more opinionated and less democratic approach to product management, including people from all origins, sexual orientations, ages, cultures and languages; empowering managers to take sound technical decisions and developers to understand the business tradeoffs.

5. Lack Of Digital Design Education

There is a fundamental problem in universities and business schools. I have mentioned above the lack of technical knowledge in MBA curriculums, so I will not dive into that matter again. But there is another issue that is dangerously blocking the prospects of the Swiss software industry: the lack of digital design education.

Most designers coming out of Swiss schools end up working in "traditional" sectors, like newspapers, printed media, advertising, while the software industry is left behind with most of the UI and UX design done by software engineers – people who, needless to say, have received virtually no education in those fields whatsoever.

For a country that has a font named after it[42], let us be very clear: it is a shame.

We need to have Digital Design curricula in both technical schools, for engineers and software developers, and also in design schools, to bring new designers in our industry. Without this injection of fresh ideas and new trends, the Swiss software industry will always be lagging behind the innovation seen in other parts of the world.

[41] /blog/developers-learn-to-say-no/
[42] https://www.hustwit.com/helvetica/

Swiss software makers must drop hierarchies, to enable a dialog between design and engineering, that is between managers, designers and developers; using a more opinionated, modern and less democratic approach to product management, including people from all origins, sexual orientations, ages, cultures and languages; empowering managers to take sound technical decisions and developers to understand the business tradeoffs, with the help of skilled digital designers innovating and setting trends.

6. Excessive Quest For Perfectionism

If there is a typical Swiss trait, is the eternal quest for perfection. Comic book readers might remember "Astérix chez les Helvètes"[43]where the Swiss prisoners in the palace of the Caesar in Rome were cleaning up everything, continuously, driving nuts the Roman intelligentsia.

Swiss people genuinely want to do their best, in every situation and every single time. As laudable as this attitude is (and, believe me, it is absolutely delightful to live in a country like this) there is a darker side as well, particularly visible in the Swiss software industry.

This tendency to perfection makes Swiss products to suffer from a major flaw: **to be late to market.** Swiss software developers have a really hard time with the concept of the "MVP" (Minimum Viable Product) and I have witnessed many disasters happening around companies literally going out of business, having no product to sell until it is very, very late.

Swiss software development teams must learn to let go, and show their work even if it does not fully satisfy the requirements for quality that they have been culturally taught to provide. It is OK to ship a not-so-good product to a smaller audience, but only if this allows you to quickly get feedback from the market, and also if you continue the development and improve the quality of that product continuously.

Swiss software makers must drop hierarchies, to enable a dialog between design and engineering, that is between managers, designers and developers; using a more opinionated, modern and less democratic approach to product management, including people from all origins, sexual orientations, ages, cultures and languages; empowering managers to take sound technical decisions and developers to understand the business tradeoffs, with the help of skilled digital designers innovating and setting trends; putting their products on the market as soon as possible to gather feedback, and using that feedback to make better products.

[43]https://en.wikipedia.org/wiki/Asterix_in_Switzerland

7. Lack Of Proper Risk Management

Switzerland is the home of the biggest insurance companies in the planet. This country has built a reputation for having the best risk managers of the world, evaluating the odds that things might go wrong, and helping manage and hopefully mitigate the risks related to any activity.

But this same country is also behind in terms of risk management for software development projects. We all know that evaluating risk in software projects is a tricky issue, yet it is also true that the lack of proper metrics from software projects blocks risk management experts from evaluating the risks properly, to create mathematical models to help us understand the odds for a particular project to go south.

The only "kind of" risk management used in Swiss software companies is actually the worst: not to do anything at all. Immobility. Projects and features regularly drop from Kanban boards all over the country because of fear, uncertainty and lack of vision.

We need insurance companies in Switzerland to start gathering data about software development projects – but this, of course, requires technical education for the MBAs that run those companies, and so far, we have none. We also need software developers and most importantly their managers to understand risk management. This will help them create better budgets and better planning, to reduce the costs of software projects in the long run, instead of choosing immobility and fear as an option.

Again, we need a dialog between business and technical teams, in order to lower the costs and increase the chances of success. Switzerland has good developers and world-class risk management staff, yet completely lacks of any kind of dialog between these two industries.

I dream of the day when the Swiss insurance industry will propose standard contracts to manage liabilities in software projects; if we had proper statistical data from past projects (both failures and successes,) and if risk management experts understood technical issues, we could model better outcomes for our projects and bring an unprecedented level of confidence in the Swiss software industry. This is yet another unique opportunity for disruption, one that could have incredibly positive outcomes for the industry.

Swiss software makers must drop hierarchies, to enable a dialog between design and engineering, that is between managers, designers and developers; using a more opinionated, modern and less democratic approach to product management, including people from all origins, sexual orientations, ages, cultures and languages; empowering managers to take sound technical decisions and developers to understand the business tradeoffs, with the help of skilled digital designers innovating and setting trends; putting their products on the market as soon as possible to gather feedback, and using that

feedback to make better products; exposing standard metrics of success and failure, enabling risk management experts to provide models to mitigate the impact of failed projects and to increase the competitiveness of the Swiss digital economy.

8. Excessive Processes Considered Harmful

The final item in this list is the one that hurts the most, and is a consequence of the many other problems I enumerated before.

Processes are the golden rule, the bureaucracy created in order to comply with hierarchies, control structures, committees, and to preserve the status quo. Switzerland lives and breathes through processes and regulations. And this is also true in the workplace. Oh, yes.

Many Swiss companies still insist in dictating to their engineering teams how to dress, what times to work, where to work from, which computers to use or not to use, which languages to use or not to use, and what kind of open-space office to setup. I have seen companies where even the layout of the furniture, the location of the whiteboards and the size and height of desks are completely described and setup by ad-hoc teams.

Do not get me wrong; some of these regulations are a godsend. For example, fire regulations. Swiss security guidelines are extremely strict, yet at the same time superbly easy to follow. Switzerland has an incredible low rate of deaths in fires and related accidents because they are that good at teaching, preventing and communicating about risks.

But it all goes overboard very quickly. Do not be surprised if…

- … a CTO explicitly forbids the use of open source code inside of an organization.
- … an IT department prevents a consultant from connecting a Mac to a wifi network in order to create the iOS application he has been hired for.
- … the same IT department cuts the access to a cloud-based continuous integration server without warning, and then threatens someone of termination because of a “security breach.”
- … an employee get a sermon by her HR manager because of the use of a short-sleeved shirts in summer without authorization.
- … a woman is fired by that same HR manager after coming back from pregnancy leave and just gave birth to her baby (yes, that can happen[44] in Switzerland.)
- … consultants are asked to deliver the final software to their client in a USB stick, because FTP or Dropbox are blocked because of security concerns.
- … a manager yells at her subordinate on the phone because she forgot to enter data in your three separate timesheet systems at

[44] http://lenews.ch/2016/05/19/too-many-mothers-are-fired-after-returning-to-work/

the end of the day.
- … one can hear laughs when asking for a VPN access to be able to work from home.
- … engineers have to wait for two months to have a server installed in the DMZ of a corporate network.
- … a manager breaks a four person team into two subteams of two people, with one “leader” each, and then enters that information in the local Active Directory of the company so that he can email only to the two “subteam leaders.”

All of these situations are the outcome of the rigidity of procedures, all going against the productivity and even the well-being of people actually performing their day-to-day tasks.

I know no HR manager working in a Swiss software developer company who has read the book Peopleware[45]. I hope some have, though. They are supposed to be “managing” software engineers, developers and testers after all.

We still pretend to have a serious “software industry” in the country, when we actively dismiss the core values that make great software teams: **flexibility, trust & communication.** One cannot replace these three values with hierarchies and control systems.

And speaking about HR, let us not get started on hiring processes; they are the worst, geared not to find talent but to fulfill a checklist of requirements; unless all the checkboxes are ticked, HR managers will never call you or even acknowledge your application. They also fail to tackle the issues I talked before about diversity, harassment, burnout, turnover and suicide in the workplace, as if these problems did not exist at all.

We need to focus less in processes and more in the people doing the work. Companies are about people, not about processes. We need HR managers and teams to understand that software workers do things differently, and that the traditional management systems used in Switzerland since the nineteenth century are completely out of context in the digital world of the twenty-first century.

Taking this into account, the final version of our manifesto for a better Swiss software industry looks like this:

Swiss software makers must drop hierarchies, to enable a dialog between design and engineering, that is between managers, designers and developers; using a more opinionated, modern and less democratic approach to product management, including people from all origins, sexual orientations, ages, cultures and languages; empowering managers to take sound technical decisions and developers to understand the business tradeoffs, with the help of skilled digital designers innovating and setting trends; putting their products on the

[45] https://en.wikipedia.org/wiki/Peopleware:_Productive_Projects_and_Teams

market as soon as possible to gather feedback, and using that feedback to make better products; exposing standard metrics of success and failure, enabling risk management experts to provide models to mitigate the impact of failed projects and to increase the competitiveness of the Swiss digital economy; finally, centering the focus of attention on the workers, those designing, creating and maintaining the systems that bring value to companies and society as a whole.

A New Hope

I have seen, thankfully, some companies breaking the mould: I can mention Ubique[46], Liip[47] and Antistatique[48] in this group. These companies are actually breaking the rules. They really "think different" when it comes to making apps through engineering and design. They break hierarchies. They setup their offices to increase happiness. They are diverse. They earn awards for their work. They thrive in a very competitive domain. And it shows in the incredible quality of their applications and systems, most of which is sadly invisible to end users, but shines through their speed, usefulness and performance.

(And for those wondering, I am not affiliated with neither of these companies, now or in the past. I just happen to know personally some people in their teams and I have seen how they work. They are awesome, and more than you think. There might be more companies like these I have not heard about – I hope!)

Please pay attention to the fact that I have left out of the equation some classical arguments, namely the "excessive costs" of Swiss manpower, or the "lack of venture capital." In my opinion neither is a deciding factor for the lack of brilliance in the Swiss software industry. First, I have personally seen incredibly effective teams working in Switzerland, with Swiss salaries, in Swiss locations, renting Swiss offices, scrupulously obeying Swiss laws. And secondly, money is not a rare commodity in this country; lots of venture capitalists live and work here. I will leave the economical argument to feed the headlines of financial magazines.

I will not include in this analysis big corporations, such as IBM, Yahoo! or Google, who happen to launch "research labs," startup incubators or just administrative offices in the country. My personal impression is that these big companies are primarily present in Switzerland to avoid paying excessive taxes in their home countries.

[46] http://ubique.ch/
[47] https://www.liip.ch/
[48] https://antistatique.net/

Conclusion

The point of this article is that the biggest challenge faced by Switzerland to thrive in the software industry has to do with the social dynamics required in software development teams. Or, at the very least, to unmask the riding hypocrisy that populates the tech sector in this country. Let us not pretend that we are better than we really are; the Swiss have many qualities, but software engineering clearly is not.

But it could be.

I would also like to clarify one thing: **Swiss engineers are individually brilliant; the problem is the lack of a large number of brilliant teams.** Think about the Argentine national football team[49]; it featured for the past 25 years the most expensive and skilled players in the planet, yet it failed to win a single international championship (with the exception of the gold medal in the Olympics.) The situation here is the same. **The engineers are, indeed, great; the typical Swiss team dynamics are horrendous.**

For Switzerland to become a world-class hub for software developers, some attitudes must change substantially, in order to let go some "traditional Swiss values" that do more harm than good in the software economy of the 21st century.

I observe that the biggest problem for setting up a successful software development hub is not the lack of venture capital, or the position in the Big Mac Index[50], but rather the cultural values that shape the group of people trying to create a software product together. One thing is to tackle the problem of finding and hiring talent; the one nobody is tackling in Switzerland is the problem of keeping talent from leaving. We still view software developers as interchangeable parts.

The good news is that I have seen younger software companies embrace the cultural change, from Geneva, Lausanne and Fribourg to Bern, Basel and Zurich, and that there is hope for the future. By mixing flexibility and trust into a culture of quality engineering, Switzerland could become a major power in the software economy. And the change is happening, right now.

Software is a social process, and as such, it has its own laws, in many ways disobeying those of physics and sociology. A large majority of Swiss entrepreneurs have failed and still fail to open their eyes, to realize that a new economy requires a new social contract. I hope that these lines, written from the trenches, will help driving a much needed change.

[49] https://en.wikipedia.org/wiki/Argentina_national_football_team
[50] https://en.m.wikipedia.org/wiki/Big_Mac_Index

Refactoring iOS Projects

Adrian Kosmaczewski

2016-07-17

Presentation given in Dnipropetrovsk, Ukraine, on July 16th, 2016. In this session we are going to learn simple yet effective techniques to refactor large iOS codebases in order to make them more testable, to adapt them to be eventually rewritten in Swift, and to make them as "future proof" as possible.

Hi everyone, thanks for choosing to attend my session and thanks to the organisers of the conference for the opportunity to talk about this subject.

So, refactoring iOS projects. Refactoring is available in Xcode on the `Edit` menu, and also when you right-click on the source code in the editor.

Thank you so much for your attention.

Oh, sorry.

Of course, we all know that AppCode is much better for refactoring. Actually it is the number one feature that every AppCode user ever mentions to me when they evangelize AppCode. It is so good that there is even a page in the JetBrains website[1] that talks about it!

In that page you can watch the difference between both IDEs.

Seriously?

Is that all? Is refactoring just a menu entry in an IDE? Is it just a marketing thing? Do we choose IDEs because of refactoring? Do we choose programming languages because of the IDEs? Do we choose the platforms we love because of the programming languages?

Well, yes and no. There is a lot of both, actually. To be a developer is also to wear the t-shirt of your favorite operating system, programming language and IDE of choice.

For example in my case it is macOS, C++ and Vim. But each one of you will have their own preferences.

[1] http://www.jetbrains.com/objc/features/refactorings-and-code-generation.html

Big Topic

Refactoring is a big topic indeed. While I was searching material for this presentation, I came across a rather large number of books with the title “Refactoring” in it, and clearly you can find a refactoring book for pretty much any programming language these days.

But in my view the most important of all of these is the original book by Martin Fowler, originally published in 1999 and that I am sure you have a copy in your own bookshelf, even if you never took the time to actually read it.

Refactoring became an industry, actually, and during the first few years of the 2000s we saw a family of “refactoring” books, first a few… then a lot…

So I decided to go back to the roots of this book, originally written with Java code examples. See, Java was the Swift of 1999, in every sense of the term, because even IBM was using it everywhere just like they are using Swift now.

But at the same time it has not, and we still need to refactor.

What is Refactoring?

The official definition of Refactoring is in page 53 of the original book:

> Refactoring (noun): a change made to the internal structure of software to make it easier to understand and cheaper to modify without changing its observable behavior.

From the noun we have also a verb, “to refactor.”

Why Refactoring?

There are several reasons to refactor software:

- To improve the design of software
- To make it easier to understand
- To make it easier to find and fix bugs
- To program faster

When to refactor?

Do not set time aside for refactoring: just do it.

- Refactor when you add features
- Refactor when fixing bugs
- Refactor during code reviews

But do not refactor when you agree that you should rewrite from scratch. Sometimes code is not even worth of a refactor. Of course, remember Netscape before doing a full rewrite of your software.

Bad Smells

The original Refactoring book has a long list of “typical” bad smells, mostly discovered and based on Java or C++ code, that can be corrected by refactoring techniques. Here is the full list of these refactoring, as catalogued by Fowler:

- Duplicated code
- Long methods
- Large class
- Long parameter list
- Divergent change or Shotgun surgery
- Feature envy
- Data clumps
- Primitive obsession
- Switch statements
- Parallel inheritance hierarchies
- Lazy class
- Speculative generality
- Temporary field
- Message chains
- Middle Man
- Inappropriate intimacy
- Alternative classes with different interfaces
- Incomplete library class
- Data class
- Refused bequest
- Comments

Requirements for Refactoring

Testing. The most important thing to know about refactoring is that there is no point on doing any refactoring if you have not taken the time to write any unit or functional tests around your code.

So if your projects do not have extensive unit testing, I suggest that you stop everything that you are doing, and that you add them to your project before doing any refactoring. It is an order. And if your manager tells you that there is no time for that, that the customer does not pay for that, that the project is small enough, that this is just a prototype, and prevents you from writing tests, well then change jobs, because you are in the wrong place.

Unit testing and refactoring are the basic pillars of any valid QA strategy, and I know by experience that non-technical project managers very often fail to understand that developers must do these tasks, and that they are as important as writing the code itself.

Specific iOS Smells

The original Refactoring book uses Java for all of its code examples. Java is quite a rigid language, much more than Swift anyway. In particular it has only recently included functional features (lambdas exist only since Java 8) so at the time the book was originally written, it was quite a pure object-oriented language only.

In my professional life, I often had to review code of other iOS developers, and in some cases I had to help some of these developers to finish their projects and clean up their code. I have done this regularly since at least 2009 and I can positively say that I have seen a bit of everything. This talk will be a short summary of some common problems I have seen in iOS projects, and some simple measures to solve those problems. The idea is to make iOS projects future proof at any given time.

In the code samples of this presentation I will use Swift 3.0 on Xcode 8.

Some typical smells in iOS projects are the following, grouped as follows:

- Programming language smells.
- Class design smells.
- Project management smells.

1. Programming Language and Cocoa Smells

- SwiftyLeaks
- Hungarian Notation
- Main Thread Fatigue
- Objective-C Nostalgia
- Homemade Cache
- Tagged View
- Illicit Association
- Long Method Names

1.1 SwiftyLeaks This is a very important point, so I am going to go through it briefly just to get the idea straight for everyone.

Programming languages are nothing else than the description of a series of mutations in a memory space, executed by a CPU. These mutations are carried by operating system processes. Processes in iOS, just as in most modern operating systems (yes, this is true from Windows to Unixes to Android to...) divide the memory allocated in several different segments:

- The TEXT segment contains the binary of the application, running in memory.
- The DATA segment contains all the static variables you have declared in your code.

- Each thread of execution has its own STACK; this segment contains the local variables and is divided in “frames”; every time you enter a function or call a method, the CPU “push” a frame on the stack, where all the local variables are created. After the `return` statement, the frame is “popped” and all of its contents are lost. This is the reason why in C++ one does not return a pointer or a reference to a local variable; they will not exist after the method exits. In Objective-C, not only values exist in the stack, but also `struct` and blocks. In C++ you can create objects on the stack just as you would create any variable.
- Finally, the HEAP segment, shared among all threads, contains all the memory that is allocated using `malloc()`, `calloc()` and `ralloc()`. All Objective-C objects created using `[[ClassName alloc] init]` exist in the heap exclusively. In C++ you can create objects on the heap using the `new` statement. You reference objects in memory on the heap using “pointers” (and “references” in C++, which are a special type of pointer that simulates value semantics) and of course, if you lose the pointer, then the memory on the heap can never be reclaimed, and we have this situation known as a memory leak.

In Objective-C, blocks are the only type of object created on the stack. They are automatically and transparently copied to the heap as soon as they are returned from a function, and then they are passed around as references.

In Swift the situation is less similar to Objective-C and more similar to Java or C#. Value types like primitives and `struct` could be allocated on the stack, and reference types could be allocated on the heap, but they could end up on another segment for performance purposes. This is something that the runtime manages completely for you. The Java JVM[2] and the .NET CLR[3] do these kind of optimisations in the background as well, and usually with excellent results that you get for free. Read more about this[4].

The important thing to know here has to do with language interoperability. If you still need to use Objective-C code in your future applications, remember that it allocates objects on the heap with the sole exception of blocks. In the case of Swift, `enum` and `struct` types are allocated on the stack.

In Swift, there are only two big reasons to consider if you need to choose between `class` and `struct`; choose classes if:

- You need inheritance
- You need reference semantics to share state
- You need to use Swift types in Objective-C.

[2] http://programmers.stackexchange.com/a/263542/741

[3] https://blogs.msdn.microsoft.com/ericlippert/2009/05/04/the-stack-is-an-implementation-detail-part-two/

[4] https://www.mikeash.com/pyblog/friday-qa-2016-01-29-swift-struct-storage.html

In all the other cases, use structs. Just for the sake of code interoperability, please find below the features of Swift not available in Objective-C:

- Tuples
- Enumerations
- Structures
- Top-level functions
- Global variables
- Typealiases
- Swift-style variadics
- Nested types
- Curried functions

This list originally included Generics too, but they have been added to Objective-C a bit later after the appearance of Swift.

1.2 Hungarian Notation This is a very common problem, and the source of endless jokes. You can tell that a C# developer has been writing Objective-C because method names start in uppercase, or that a Java developer was writing Swift because there is an abstract factory for every single class in the system.

Please pay attention to the naming conventions of the language you are using, and adopt them. This is useful for many reasons:

- It helps in the long term for the maintenance of the software.
- It helps new team members read the existing code.
- If your system ever ends in the open source domain, other people out there will have less trouble understanding your code.

  Programs must be written for people to read, and only incidentally for machines to execute.

Harold Abelson, “Structure and Interpretation of Computer Programs”

So here go some useful links:

- Cocoa naming guidelines[5]
- Objective-C naming guidelines[6]
- Swift API design guidelines[7]

Particularly for Swift 3, the main guideline to remember is to **Make uses of your APIs read grammatically:**

```
friends.remove(ted)
mainView.addChild(button, at: origin)
truck.removeBoxes(withLabel: "WWDC 2016")
```

[5] https://developer.apple.com/library/mac/documentation/Cocoa/Conceptual/CodingGuidelines/Articles/NamingMethods.html

[6] https://developer.apple.com/library/ios/documentation/Cocoa/Conceptual/ProgrammingWithObjectiveC/Conventions/Conventions.html

[7] https://swift.org/documentation/api-design-guidelines/

Also name methods based on their side effects: use verbs to describe the side effects:

```
friends.reverse()
viewController.present(animated: true)
```

And use nouns to describe the result or what is being returned:

```
button.backgroundTitle(for:.disabled)
friends.suffix(3)
```

Also, pay attention to the "mutating / non-mutating pairs" of methods:

```
x.reverse() // mutating
let y = x.reversed() // non-mutating

dir.appendPathComponent(".txt") // mutating
let newDir = dir.appendingPathComponent(".txt") // non-mutating
```

Starting in iOS 10, every API will have two names:

1. One for Objective-C & Swift 2
2. Another for Swift 3

Swift 3 APIs can be annotated with the `@objc()` attribute, specifying a method name to be used with Objective-C.

Inversely, when exporting Objective-C APIs to Swift 3, you can now use the `NS_SWIFT_NAME()` macro to specify the names that you want to use in the newest version of the language.

Please watch session 403 of WWDC 2016 for more information about this subject.

1.3 Main Thread Fatigue This is a common mistake but relatively easy to solve these days, primarily thanks to GCD.

Four things to keep in mind:

- Separate Core Data MOC for background queues and threads
- Number crunching, filters, etc, use GCD
- Network operations, use the `NSURLSession` class family.
- Main thread === UI thread. Nothing else. Everything that cannot be seen should happen on a background GCD queue.

Pay attention to the new syntax of GCD APIs in Swift 3!

1.4 Objective-C Nostalgia You should seriously start thinking about migrating your Objective-C code to Swift in the near future. Swift 3 is going to be ABI compabible with Swift 4 (that is at least the explicit wish of the Swift team at Apple) so you should start adapting your Objective-C code so that it can be used by Swift classes.

- Use the `nonnull`, `nullable` and similar declarations
- Use generics for your containers:

```
NSArray<NSString *> *arrayOfString = @[@"one", @"two"];
```

Beware of the following caveats during the migration process:

- Objective-C cannot subclass a Swift class, unless it is marked with the `@objc` decoration.
- Tuples, Swift enums and structs are not accessible from Objective-C.

Objective-C will remain in my opinion a very useful language to wrap old C APIs and libraries, in order to make them accessible from Swift.

A special comment about the use of `#define` in Objective-C. This is a pet peeve of mine. Instead of

```
#define RADIUS 45.5
#define NAME_DICT_KEY @"name"
```

Do this:

```
static CGFloat RADIUS = 45.5;
static NSString * const NAME_DICT_KEY = @"name";
```

The above declarations only work when the symbols are used in the same compilation unit (read, `.m` file) where they are defined; if you need constants to reuse, do this:

In the header file:

```
NS_EXTERN NSString * const NAME_DICT_KEY;
```

and in the implementation file:

```
NSString * const NAME_DICT_KEY = @"name";
```

In Swift, needless to say, you can use nested enums inside of your structs and classes to define your own constants.

1.5 Homemade Cache Over the years I have seen a lot of applications that feature some kind of homemade cache object, to store images downloaded from the network or other artifacts. Creating these objects is really complex and requires lots of attention and testing… and you do not need to do that.

Just use `NSCache`. As easy as that. The problem with `NSCache` is that it has a relatively complex API documentation, so it is sometimes complicated to know which options to pass to the object initializer.

Fear no more, the code for a nice and simple instance of `NSCache` is here.

```
let config = URLSessionConfiguration.default()
config.requestCachePolicy = .returnCacheDataElseLoad
let memoryCapacity = 10 * 1024 * 1024;
let diskCapacity = 20 * 1024 * 1024;
let cache = URLCache(memoryCapacity: memoryCapacity,
                     diskCapacity: diskCapacity,
```

```
                          diskPath: nil)
URLCache.setShared(cache)
```

1.6 Tagged View There used to be a time when this was acceptable code:

```
override func tableView(_ tableView: UITableView,
                        cellForRowAt indexPath: IndexPath) -> UITableViewCell {
    let cell = tableView.dequeueReusableCell(withIdentifier: "Cell",
                                             for: indexPath)

    let object = objects[indexPath.row] as! NSDate
    cell.textLabel!.text = object.description

    let switchControl = cell.viewWithTag(1) as! UISwitch
    switchControl.setOn(false, animated: false)

    return cell
}
```

This is no longer acceptable. Please do not use tags like this. It leads to absolutely unmanageable code and it can lead to weird bugs. Particularly if your code uses magic strings and numbers all over the place.

And please do not use Swift `enum` to replace those tags. Just do not use tags.

```
override func tableView(_ tableView: UITableView,
                        cellForRowAt indexPath: IndexPath) -> UITableViewCell {
    let cell = tableView.dequeueReusableCell(withIdentifier: CellIdentifier,
                                                                  for: indexPath)

    let object = objects[indexPath.row] as! NSDate

    cell.titleLabel.text = object.description
    cell.switchControl.setOn(true, animated: false)

    return cell
}
```

And the definition of the `CustomViewCell` could not be simpler:

```
class CustomViewCell: UITableViewCell {
    @IBOutlet weak var switchControl: UISwitch!
    @IBOutlet weak var titleLabel: UILabel!
}
```

And you should of course make the connections in Interface Builder, as required.

1.7 Illicit Association Associated objects are a great thing.

```swift
import Foundation

class SomeObject : NSObject { }

extension SomeObject {
    private struct AssociatedKeys {
        static var AssociatedValue = "AssociatedValue"
    }

    var associatedString: String? {
        get {
            return objc_getAssociatedObject(self,
                                            &AssociatedKeys.AssociatedValue) as
        }

        set {
            objc_setAssociatedObject(
                self,
                &AssociatedKeys.AssociatedValue,
                newValue as NSString?,
                .OBJC_ASSOCIATION_COPY_NONATOMIC
            )
        }
    }
}

var obj = SomeObject()
obj.associatedString = "Some other string"
print(obj.associatedString)
obj.associatedString = nil
print(obj.associatedString)
```

1.8 Long Method Names Old Objective-C APIs, when translated verbatim into Swift, will inevitably look verbose. Think about renaming them.

2. Class Design Smells

- Massive View Controller (MVC)
- Massive App Delegate (MAD)
- Forgotten Memory Warnings
- Long Switch Statement
- Excessive Curiosity
- Selfish Navigation
- Offline Confusion

2.1 Massive View Controller (MVC) This is by far the biggest problem I have seen in large iOS projects, one that I have been the

culprit of causing several times, and I am pretty sure you have had this problem too in your own code at least once.

`UIViewController` is a fundamental piece in the UIKit framework. It can do everything, and sometimes it ends up doing too much. This situation must be prevented at all costs, because it makes very difficult to maintain large iOS applications – and all applications end up being large at some point.

Several strategies to break up controllers:

- Categories: break up your controllers in separate categories with distinct responsibilities.
- Helper objects: break up your controllers and create private, helper objects that take care of different functionalities.
- State pattern: if your controllers must act differently depending on their state, do not write `if` statements for that! Never, never, never. Use the state pattern. This requires you creating a separate class for each different state of your controller, and then perform the changes inside of those state subclasses. The GameplayKit framework contains since iOS 8 a very handy `GKStateMachine` class, with a matching `GKState` class, that you can subclass, and it offers a wonderful infrastructure for a very simple and straightforward implementation of the state pattern.

For those looking for a simpler approach, you can use a very simple Swift enum that takes a block as its associated values, and then you just execute the associated blocks. This provides a simple approach that allows you to organise your code.

The objective of breaking controllers down must be that the files defining them are never bigger than 400 lines of code. This is already a sizeable amount of code for a class, but it is manageable.

Check out this blog post[8] for more alternative architectures.

2.2 Massive App Delegate (MAD) This problem, somehow related to the previous one, is the result of several factors:

1. Using Apple templates, particularly when selecting the "Core Data" option in Xcode
2. Not knowing what the App Delegate is there for

The application delegate is an ear. It listens for events from the application object, many of which come directly from the operating system itself. The idea of the app delegate is that it exists only so that you can have a centralized location for all the events that come from the operating system during the lifetime of the application. **And nothing else.**

This way your code knows when your application has started, with what parameters, when it is going to be sent to the background, when

[8] https://swifting.io/blog/2016/09/07/architecture-wars-a-new-hope/

it is going to be killed, when there is an incoming memory warning, and things like that.

All code that does not explicitly pay attention to these events should not exist in the App Delegate. Please pay attention to this fact, and create separate objects to take care of different concerns.

This is a basic design issue, one that can be solved easily, yet I keep seeing it everywhere. My personal solution would be to avoid the default templates of Xcode, unless you are building a proof-of-concept or a prototype. Big projects should start as minimalist as possible, simply selecting the "Single View" template.

2.3 Forgotten Memory Warnings There are three types of memory warnings:

1. The App Delegate features the `applicationDidReceiveMemoryWarning(_ application:)` method.
2. Every `UIViewController` subclass should override `func didReceiveMemoryWarning()`
3. Finally, any object can listen to the `UIApplicationDidReceiveMemoryWarning` notification and respond accordingly.

But the sad truth is that most apps just choose to ignore all of them. Come on, we have all been guilty about this. And of course we can argue that the latest devices have more and more RAM. Of course the latest iPad Pro boasts an incredible 4 GB of RAM, so we might think that we just do not need to care about this anymore.

False.

Listen to all of these notifications. Implement all the methods. Clean any data that can be re-calculated later on, save files, bring down any complex data structures you will not need in the short term and that can be recreated or reloaded later. iOS still sends memory warnings, even in devices with lots of RAM, because memory management. And one cannot know exactly when or why this is going to happen, but be sure that this is going to happen sooner than later.

Lazy-loading is your friend. You can implement a public calculated property wrapping a private field, which has the advantage over `lazy var` statements in that their contents can be cleaned up at any moment.

2.4 Long Switch Statement This is a smell that was identified by Fowler in his book, but it is so prevalent that I would like to spend some time discussing some solution for this smell.

Long switch statements are drag. You all know that. However, for one reason or another, we keep doing them. Of course we are very happy now that Swift implements a safe `switch` statement, one in which one has to actually write the `fallthrough` keyword to jump from case to case.

On the other hand, Swift brings so many niceties to `switch` statements, such as pattern matching, that it is very tempting to use them and to use the often.

But they are drag for several reasons:

- They increase the number of paths in the code, making it harder to test.
- They can decrease the readability of the code when they span several screens.
- They can contribute to the MVC smell (Massive View Controller.)

Three strategies to solve this problem:

1. Extract methods for each switch statement first.
2. Move those methods to separate objects.
3. Remove the switch statement altogether with polymorphism
4. Use the state, decorator or strategy patterns for more complex situations.

Using these strategies you will ensure that your code is maintainable and testable.

2.5 Excessive Curiosity If you see any of the following in a code, this is a rather strong smell:

```
// Only if you need compatibility with iPhone OS 3.2... really?
if UI_USER_INTERFACE_IDIOM() == UIUserInterfaceIdiom.pad {
    // ... iPad-only code
}


let idiom = UIDevice.current().userInterfaceIdiom
if idiom == UIUserInterfaceIdiom.phone {
    // ... iPhone-only code
}
```

To begin with, you should encapsulate iPad-only or iPhone-only code in universal apps in a separate class, and test it accordingly. This includes controllers, helpers, even models that might run in another context. If the amount of code is too much, you might want to separate it in its own framework, but again, pay attention to testability and simplicity.

If you still want to do version checking, since iOS 8 you can use `NSProcessInfo`[9] like this:

```
// Since iOS 8:
let version = OperatingSystemVersion(majorVersion: 10, minorVersion: 0, patchVe
if ProcessInfo().isOperatingSystemAtLeast(version) {
    print("iOS >= 9.0.0")
}
```

[9] http://nshipster.com/swift-system-version-checking/

```
let os = ProcessInfo().operatingSystemVersion
switch (os.majorVersion, os.minorVersion, os.patchVersion) {
case (8, 0, _):
    print("iOS >= 8.0.0, < 8.1.0")
case (8, _, _):
    print("iOS >= 8.1.0, < 9.0")
case (9, _, _):
    print("iOS >= 9.0.0")
default:
    print("iOS >= 10.0.0")
}
```

But try not to. In the same category I would mention other similar mechanisms, such as `if #available(iOS 9, *) {}` or `[[NSProcessInfo processInfo] operatingSystemVersion]` or `[[UIDevice currentDevice] systemVersion]`. Just don't.

2.6 Selfish Navigation I have seen many apps using a single controller, and performing all visual changes inside of that controller, using animations, lots of views, and a severe case of MVC (Massive View Controller.)

This is wrong. The whole idea is that every screen in your application, as designed by your UI and UX teams, must map to one and only one subclass of `UIViewController`.

Of course in the case of the iPad, if you are using popover controls, you are most probably going to have lots of small view controllers. In that case remember that view controllers can be nested.

And to animate the transitions between those view controllers, you can define your own custom transitions to make them look any way your designers want.

2.7 Offline Confusion Making applications that store data locally and also load data from the network can be tricky, so here goes a simple pattern that works for every application that has to load data locally and from the network as well:

1. Start your application by loading the UI using data from the local data storage. Do not block the user.
2. If no local data available, inform the user with a "blank" state screen.
3. While this happens, start async network operations and download the data.
4. Save the remote data locally.
5. Notify the UI to refresh itself.

If you use NSCache then step 3 will return immediately, but your view logic should not know anything about this fact. Follow the steps above and in that order.

3. Project Management Smells

- Invisible Documentation
- Template Abuse
- Folder Clusterfuck
- Cocoapods Galore
- Rogue Compiler Warnings
- User Discrimination
- Interface Builder Attack
- Framework Infection
- iOS Nostalgia
- Backend API Anarchy
- Chatty API
- Splash Screen

3.1 Invisible Documentation I somehow feel bad having to say this, but seriously, this one is a constant in the industry. Look at the following (wrong) excuses to not to write documentation:

- We are agile
- Documentation is always out of date
- We haven't budget for that
- Nobody reads it anyway
- Our code is self-documenting
- Real developers don't need documentation

I say **bullshit**. If you have not written documentation, you have not done your job. And this applies not only to code that we write for ourselves, it applies particularly for the code that we write for others.

I will confess that I myself I am sometimes late at writing documentation; if you look at my project SwiftMoment[10] in Github, you will see that I have an open issue to write documentation[11]. That is (among others) the reason why this library has not yet achieved version 1.0 (it will, soon.)

What must be documented?

- Public classes, methods, enums, properties, everything that could be exposed to the outer world.
- Storyboards and XIBs: where are the UIs of the view controller classes defined, and how?
- Xcode schemes.
- Testing infrastructure setup (Jenkins, Xcode bots, etc.)
- Build setup procedures, including deployment information to the App Store or other signing issues, and how to solve them.
- Hidden dependencies: libraries, operating system versions, etc.
- Code that has been cut-and-pasted into the application, citing sources (URLs, books, etc.)

[10] https://github.com/akosma/SwiftMoment
[11] https://github.com/akosma/SwiftMoment/issues/58

- Notifications and KVO use.
- Workarounds for bugs in external libraries of pods.
- Branching strategy for the repository.

How should these things be documented? The acronym representing the three pillars of code documentation is “ARW”:

- API documentation in the code.
- README file at the root of the project.
- WIKI with more information, as required.

What should not be documented?

- Private members and classes.
- External libraries and pods.

Regarding `NSNotification` and KVO:

Notifications are great. They allows to decouple classes from one another and to reuse them in other contexts. They bring great power.

However, with great power comes great responsibility. Common problems I have seen around notifications are the following:

First, many developers still use plain strings for the notification names, scattered all over the source code. No. Use `static` constants, `extern` constants or Swift `enum` inheriting from String. In Swift embed the enum of notifications exposed by a class inside that same class, to increase code readability.

Second, no API help for the notifications whatsoever. This is a major no-no. Notifications are all about side effects; they introduce hidden dependencies in your code, that is, a change in one class may trigger a side effect on another. They can lead to code that is difficult to debug (hello thousand breakpoints scattered all over the code) and it makes the code really hard to maintain.

Every notification in the project should have its corresponding API documentation, explaining two basic things:

1. Who usually registers for this notification
2. When is the notification triggered and by whom

As general advice, remember that notification handlers should not trigger other notifications (that is, there not should be second- or third-order side effects) and that KVO notifications are the trickiest to figure out.

Also, I suggest that you start using `#keyPath()` in Swift 3, so that you have strong-typing compiler support for KVO observers.

3.2 Template Abuse Because we are “agile” it does not mean that we do not think anymore. For some reason we open Xcode and start pouring code even before sitting down and trying to understand the problem we are dealing with. And these days, “design” rhymes with UX and UI but nothing else.

Us, software developers, before starting to code a project, we must take a step back and think. We have to take a look at the available technologies and actually design our software using methodologies that actually make sense and convey information – and also, in the long run, become part of the documentation of the system we're trying to create.

Have you heard of CRC cards[12]? Everybody has its mouth full of agile this and agile that but the truth is that apart from the daily meeting in the morning we barely apply any other agile technique whatsoever.

Before writing the next class in your project, take a time to figure out what every class, struct, enum and lambda in your project is there for, figure out how to test each part of the system, and then move forward.

3.3 Folder Clusterfuck People: it is 2016 and I still see project folders like this.

This cannot be, and it does not have to be like this.

Organize your code in folders, any arrangement will do: maybe using the typical "Rails" structure of "Models / Controllers / Views / Helpers" or any other organisation, but please take the time to organize your code in an appropriate way.

By the way, not using image catalogs in 2016 is a shame. Your designer should be producing them for you, as a matter of fact, so teach them how to use Xcode so that they can create those catalogs for you. Your project will be much better in the future.

3.4 Cocoapods Galore Do you really need to have more than five (5) external Cocoapods in your project? Seriously, no. You do not need that many. Think about it. iOS these days brings lots of functionality that it makes up for most small libraries out there.

Pick five big projects that you really need, and then only use those pods. For example, PromiseKit, AFNetworking, OCMock and frameworks like that. They really bring value, they really help you because they offer things that iOS still does not. All the other libraries (yes, including SwiftMoment) you do not need it.

3.5 Rogue Compiler Warnings This is a favorite of mine, and it keeps happening time and again. Particularly in old codebases that have not been updated, say, since iOS 5 and such.

Leaving warnings unattended is a big problem. Just assume that compilers are made by people who know the language much better than you, and in particular, they know how the language will evolve in the future.

[12] http://www.agilemodeling.com/artifacts/crcModel.htm

Whenever you see a compiler warning, particularly when dealing with Objective-C code, you should immediately fix it. All code checked in source control should always: compile, pass tests, and have zero warnings.

In particular, Xcode 8 brings a new kind of warning to the table: "Runtime warnings", that is, situations that arise while the application is running under the supervision of Xcode. These could be harmless situation, such as autolayout incoherencies, but it could also be more complex things like memory leaks or circular object references.

A warning today, an error tomorrow. It is as easy as that. Pay attention to all of them and correct them.

3.6 User Discrimination Developers: stop releasing applications that do not include support for the incredible accessibility features of iOS!

Not only do they bring good karma points to your team and product, they are also required for UI testing in Xcode. So, please, make sure that you do support at least the following accessibility features:

- Dynamic text (so that the text in your UI changes size automatically depending on the preferences of the user.)
- Accessibility hints (used by VoiceOver to "speak out" different elements in the user interface for visually impaired users.)

3.7 Interface Builder Attack Storyboards are an interesting addition to the toolbox of iOS developers, however they bring a couple of problems to the table:

- Just like XIB files, they do not play well with source control. Have you tried fixing conflicts in Storyboard files? Not pretty.
- Impossibility to know how to reuse a controller in code. Should I `ViewController()` it or should I call `storyboard.instantiateViewController(withIden` And which storyboard should I use?

There are a couple of things that you can do to solve these problems:

1. Document the storyboards. Create a small `STORYBOARD.txt` file in your project that explains, for each storyboard file, which controllers are defined therein and what are the types of segues that connect them. Create entries for each storyboard in your application. Also: explain in the API documentation for each controller the name of the storyboard or XIB file where its UI is defined. This will make everyone's life easier.
2. Prefer individual XIB files for the UI of a particular view controller. Use the storyboards just for the navigation, but define the UI of an individual controller inside of its own XIB file. This way your controller will be more reusable in other contexts. Also, remember not to hard-wire references to other controllers in your own controller; this way you can make your controller more reusable.

3. Have separate Storyboards for iPad and iPhone. Enough said.
4. Prefer code to storyboards for large projects. Seriously. Believe me. And if you do not believe me, know that Google explicitly bans the use of Storyboards in their own iOS teams, and they do everything by code.

3.8 Framework Infection Since iOS 8 we have been able to create dynamic libraries for our projects. This is great, but at the same time I have started seeing a worring pattern: some projects divide their code in so many subframeworks, which is often an exaggeration.

This is unfortunate, because it introduces many small problems in terms of configuration and signing, and you should limit the number of subframeworks to a minimum. In particular, this technique is really useful only if you are targeting several different platforms with your code, and you want to share your code among them: tvOS, watchOS, macOS…

Otherwise, just use a good class design technique, create a flexible architecture, and just create a monolithic binary. Your build process will be simpler, and you will move as fast.

3.9 iOS Nostalgia iOS as a platform is evolving at a very, very high speed, and it has been like this for the past 8 years, and it will be like this for the forseeable future.

I know that some customers, for some reason, will insist that your code supports iOS 6 or 7 these days, but you have to tell them to stop doing that, for many reasons:

- Old versions of iOS are no longer supported and might have security problems.
- The costs of development and testing of old versions of an application are very high: you need a separate machine, old devices with old versions of the operating system, etc.
- It is very complicated to find documentation and online help to solve bugs in old versions of iOS.

The idea here is that you only support “current version - 1” at every single step of the way. This means that you have to write this down in the support documents that you provide to your customers, and that you have to start budgeting in September of every year the money required to migrate the code to the new version of iOS between June and September of the following year.

You can teach your customers that this is very important, and it will make life easier for everyone.

3.10 Backend API Anarchy This is a very important refactoring to make: if your mobile team is not in control of the backend API, then it is not in control of the mobile application at all.

The best mobile teams I know have a couple of good backend guys, usually building a proxy server between their mobile app and the backend provided by the customer. This brings many advantages:

- **Performance:** teams can tune the performance of the API that the mobile app uses, including the payload, its compression and other factors. You can also cache data, perform asynchronous refresh sessions, and deal with slow existing backends using SOAP or other clumsy RPC methods.
- **Quality:** you cannot rely in an API that is not in your control. What happens if your client inadvertently breaks the backend? Using your own proxy you can serve cached data, or at least fail gracefully.
- **Security:** you can ensure that all the communications between your mobile app and the API are encrypted and secure, using the industry's best practices, regardless of what the clients' IT team thinks.

APIs should not be chatty. To conserve battery power it is better to coalesce network operations in one call, returning all required information for a single screen in one simple step.

Also, explore new avenues using GraphQL[13] and Socket.io, in order to have better APIs.

Final Recommendations

As you can see, this talk about refactoring did not only cared about your code, but also about the infrastructure around your code, and even your own team organisation. Good, maintainable code is the result of many factors, and so I leave here a couple more suggestions for your convenience, to make sure that your application code stands the test of time.

- Always use the latest Xcode and Swift
- Use a git branching model
- Migrate your projects for the future (iOS 11, etc)
- Plan accordingly
- Learn about classic design patterns
- KISS principle. Always choose the simplest solution that gets the job done.

Thank you so much for your attention.

[13] http://graphql.org/

Elegia

Adrian Kosmaczewski

2016-08-05

Se trató una empedernida discusión entre personas sin estilo, que se eternizó durante un tiempo singularmente limitado. No queriendo o no pudiendo entretenerse de otra manera, quisimos entender de maneras desiguales las tremendas riñas que nos agarraban con la mano.

Como decía aquel autor, no importaba el lugar ni el momento. Podría haber sido en la Roma Antigua, en Constantinopla o en Nueva York durante la prohibición. Digamos que fue aquí cerca, no hace mucho.

En todo caso no fue así, no nos detuvimos en esas impresiones, ni quisimos dejarnos ir en abluciones mentales. No, mucho peor; quisimos destripar la palabra y dejarla fluir, de tal manera que nos llevase tal plática a otros recónditos rincones impávidos, más inocuos, pero fracasamos indudablemente.

¿Cómo se hacían verdades a partir de tales encuentros? Se generan nuevos elementos a partir de nuestras charlas; nos dejáramos, quizás, llevar por aquellas idiotas costumbres. Tal vez en alemán, tal vez en sánscrito, quizás no existan otras opciones más que las de sumar adeptos.

Y fue así que la promoción se quedó anclada en los conceptos pretéritos de la infamia burlesca, la que no perdona, la que nos impide procrear el mundo en el que lloramos. No eran éstos tiempos ni para la maestría ni para la elucubración; en realidad, todo lo contrario. Nos debíamos mutuamente la importante necesidad del dejar de ser.

Para qué negarlo, después de todo no era tanto ni era tan poco, y ni siquiera merecía tanta atención. Pero, obviamente, nos preocupábamos y nos alentábamos mutuamente para mantener encendida la llama de aquel recuerdo efímero. Palabra rara, “efímero”, nombrando y refiriéndose a aquello por lo que tanto sufrimos.

Lloramos en silencio. Nos abrazamos, entrelazados en la trama de una memoria de mierda, un recuerdo horrible, ¿para qué? Buena pregunta, para la cual aún hoy no he encontrado una respuesta coherente.

Era así como se iniciaban los contraataques de la memoria, de aquella horrible realidad sin fondo ni atención, que se entremezclaba con la

impúdica muleta del hombre moderno. Frase larga, sin sentido, pero fue eso lo que se nos ocurrió como lo más coherente, como la prueba más activa de nuestra realidad impresionante.

Quién quisiera rememorar tales trayectos oblicuos, mirando de perfil una viñeta colgada de la pared. Un abanico, una vieja foto tomada en un parque nacional en Costa Rica, un llamado de atención en un aeropuerto ignoto, un viaje al infinito en una cinta de Moëbius sin sentido ni orientación.

Fuera de broma, me preguntaba yo, ¿quién sería el responsable en tal caso? ¿Con quién debía uno quejarse, elevar una moción de censura, para evitar tales remordimientos?

Aún no me has perdonado la afrenta del pasado, me sigues criticando y me sigues juzgando por el dolor que te causé. No te culpo ni me indigno; simplemente observo con dolor que nunca lograré demostrar mi inocencia ante este juzgado.

No quieres retirar tu frente del vidrio, me ves caminar bajo la lluvia sin mirar atrás. Sé muy bien que tus ojos se llenan de lágrimas; nuestro encuentro fue en vano. Mi palabra no sirve de nada en este lugar perdido, y vale aclarar, no hablo de ese techo que nos reunió durante la corta primavera de nuestra felicidad. Hablo de un lugar conspicuo, sin rejas en los portones, con el pasto quemado, ventanas rotas, telas de araña, donde se revive cada día el mismo momento como castigo eterno.

Me ilusiono brevemente recordando que la historia de Lucifer remite invariablemente a la de Prometeo, pero la satisfacción es de corta duración.

No me dejes, véte, toda esta sinfonía suena al unísono, una nota, una tonalidad, una clave, pura monotonía, y yo no sé qué contestarte. Cierro el portón, crujen baldosas bajo mis pies.

Sandra

Adrian Kosmaczewski

2016-08-10

(Based in a true story.)

I first saw her around September 1969, at the start of my third year of high school. She walked into the classroom one morning, the new student just introduced by our teacher.

Silence invaded the room as soon as she entered, for no other reason that she was uncannily beautiful. Slim, black hair, tanned skin, bright blue eyes. Most of us had never seen such a beautiful person so far. Aware and annoyed by our stares, she quickly walked up the alley between the desks and sat at the far end of the room, next to a friend of her who she knew from outside school.

We were together in the same class for the next two years, until I left to Europe in 1971. We never exchanged much during that time. I was the nerd of the class, buried deep in piles of books, a living version of an encyclopedia able to memorize any kind of useless fact. She was the smart, witty, pretty girl, the popular cheerleader. The one that would usually be portrayed by Megan Fox.

No, scratch that; although Sandra looked eerily like Megan, this girl became my own personal Susan Glenn, decades before the meme.

I dreamt of her, even long after I left to Europe. She was my very idea of physical beauty, that which fueled my fantasies and my desires. She became a gauge, a model to which I compared every girl I saw in the street. My male mind was blurred with appearances, fed with memories and hormone spikes.

I desired her like I had desired few women in my life. She was unattainable – and quite literally so, given the thousands of kilometers that separated us. We weren't even exchanging mail (you know, the old-fashioned kind, with stamps and envelopes.) We just weren't friends. She barely acknowledged my existence in the classroom. I was nobody to her.

Come August 1971, a plane took me and my mother to a new country, to live a new life.

Fast forward July 1977, when I returned to Mexico for holidays, my first trip back to my birthplace since I had left.

To make a long story short, in a rather unexpected sequence of events, one morning I woke up and I saw the naked body of Sandra lying asleep, next to mine. I remember staring at her that morning, still in awe and thinking that this was just a dream, one which I would probably wake up from in any minute. She was intelligent, hotter than a supernova, elegant, sensual, and much to my disbelief, eager to spend a night with me.

I grudgingly smiled and came to the conclusion that it was not a dream.

Rewind to a couple of days earlier. Sandra and myself were having a barbecue, with our old friends from school. It was a beautiful starry night in Cancún, were we had gone to spend some days in the beach.

After a couple of beers I gathered enough courage and started talking to her. God was she beautiful. Her eyes. Oh. My. God.

She kept staring at me silently. I suddenly noticed that she was not drinking. I told her (in French) how much I was enjoying being with my old friends tonight (we were together in French class back in high school, hence the language choice.)

She answered to me in French: “je ne veux pas être ton amie.”

I shut up and leaned back in my chair. I did not understand at first. I saw her staring at me. I did not want to believe in anything, and the beer suddenly jumped to my head; I felt dizzy. I excused myself and walked to the balcony nearby to see the sea at night and catch my breath.

She came behind me. She grabbed my arm, turned me around and kissed me softly, quickly. Then she walked away.

The scene appears to me now as if it had been a dream. I do not remember going to bed that night.

The next morning we went to the beach, just like every other day, and she kept silent watching me the whole time. I felt embarrassed. I did not remember properly what had happened the previous evening, so I was keeping a certain distance with her.

On the way back to the hostel, I clumsily started apologizing for the night before. Yes, I truly did that. She listened to me in silence, both of us walking slowly behind our group of friends. I was somehow expecting her to be mad at me.

As soon as our friends walked around the corner of the hostel, Sandra suddenly pushed me against a tree and kissed me. Rage, passion, strength, sweetness, silence, her body and mine.

Pure latin cliché. True story.

I stayed glued to the tree, first with my eyes wide open, and then, for the first time in my life, I let go all of my fears and embraced fate, realizing that one of my dearest and strongest childhood dreams had, for some unknown reason, become a reality.

Remember what Coelho used to say, about the Universe conspiring to help you reach whatever you want most? Well, I felt that in person. I let go all of my rationality for a few seconds, maybe a minute, and I felt lighter. It felt so good. I had never experienced that before.

We stayed glued to each other for a little while. She then opened her eyes and told me to shut up. Staring at me from centimeters away, she told me that she could not take it anymore, that she had wanted me since I had walked out of that airport. I shut up and listened, my eyebrows raised, still not totally believing anything of what had just happened there.

We kissed again.

My holidays stretched until the end of August 1977. I returned to Mexico twice that year, mostly to spend time with Sandra.

I wanted to marry her. We even exchanged rings. We talked about living together. She helped me trace plans for a new life in Mexico. She got me back in touch with being a Mexican once again.

I finally decided to go back to live in Mexico in June 1978. I was going to leave everything I had in Europe behind; a crappy job, an alcoholic mother, a botched attempt at university, and a good group of friends.

The Friday before the final flight back she called me. Her voice was broken and sad. She told me that things were not going to be the same upon my arrival. I said, of course not, because I'm not leaving anymore.

No, she said, you don't understand, I will not be there.

I stayed speechless. After a few seconds I asked why.

She told me she did not love me anymore.

I felt my feet shaking. I wanted to breathe the air of Cancún once again. A film played in my head at high speed, the first frame being her eyes in that classroom in 1969.

I did not tell anyone about this phone conversation. Not even my mother. I kept it to myself. I still do not know why.

After a dreadful weekend, Monday came and my flight, too. A few friends and my mom came to the airport to say goodbye.

One of my friends, who worked at the airport, walked me to the boarding gate. She looked at me and asked me what was going on. She told me I looked pale, as if somebody had died.

I told her everything. She put her hand over her mouth, horrified and in disbelief. “Why don’t you stay then?” she asked me.

I did not know. I felt I had to go back to Mexico anyway.

The flight was boarding. We hugged. I climbed for the second time of my life on a plane that would take me to a new life on the other side of the Atlantic.

As soon as I arrived I tried to talk her back, a few times, to no avail. She avoided me. I fell into depression. I asked myself what I said or did that triggered such a reaction.

As you can read in the previous paragraph, for years I thought this separation was solely my fault. Rather, it was both our fault. I am not perfect and I know I made mistakes. But I did not deserve to be ditched like a broken toy.

We met a couple of times in gatherings and parties organized by friends in common. She was there with her new boyfriends. Note the use of plural here.

That was painful to see, of course. I felt emasculated, it is as simple as that. Maybe it is hard to understand for women, but that is exactly what I felt. Rage and pain and the urge to kill. I understood why since the times of Troy, wars had exploded over the lost love of a woman. I understood the thousand and one Boleros singing about the one that went away.

We are not in contact anymore; I think I last saw Sandra in 1980, and we did not even talk that time.

Through her I understood many things about love and life.

I am not that different from her, actually. I just had a wrong idea of love, a twisted, romantic desire that had the shape of a never ending story, something that does not exist. She was free; polyamorous, driven by desire and sure of herself and her aura, bundled with matching beauty and a wicked spirit to go with it. But she was, dare I say, imperfect, like me, like we all are.

Maybe the lesson I should have learnt by now is that I too have been assertive, have desired and have found what I truly wanted, against all odds. She.

I still have shivers down my spine when thinking about her. I wonder how much I have learnt. I try to reconstruct my own puzzle every day.

I look back at that moment and figure out that maybe, just maybe, we loved each other for a few minutes against a tree in Cancún, and that only because of that, being with Sandra was totally worth it.

Developers and Politics

Adrian Kosmaczewski

2016-08-16

Some thoughts around the tensions between technical and business teams. TL;DR: I do not offer a solution in this text for political problems in software organisations; I just want this article to help developers not to feel alone in this struggle.

Most software developers I know have a hard time understanding political power. We tend to assume that technical knowledge is power—which is not false; it is, however, a limited kind of power. I understand the word "Power," in this context, as the capacity to make things happen.

The basis of all evolution in social systems is, whether we like it or not, the constant shift and balance of power. History has been written on imbalances of power. Wars have been fought because of them. Millions have died and migrated because of them. And you, my dear reader, willingly or not, working remotely or not, in open source or in closed source projects, you are bound to obey dynamics that are more often than not driven by politics.

And your employer, your local user group, your preferred developer conference, all of these organisations and many others are also driven by politics, by people in charge telling other people what to do, usually but not always after getting their support in some way. Maybe you get a salary; maybe you get some stock options; maybe you just believe in your leader; all of these actions validate and give power to the ones that take the ultimate decisions, whatever they may be.

For software developers, the conflict appears when the decisions at stake are not entirely logic, rational, and seem driven by ego and political power quotas on the roulette table. We struggle in this kind of situations. We just do not know how to react. This is a natural part of the ongoing mismatch between suits and t-shirts, between business people and technical teams. This is part of our nature.

To be fair, this situation is not new, at all. The plots of movies such as "The Right Stuff" or "War Games" drew inspiration from the impedance mismatch between technical and business teams. But in your case, dear reader, your nodding while reading this text has nothing to do with an Oscar-winning performance. You would like to understand why your project managers are getting bonuses while you

are not. You would like to know why some Scrum implementations become NSA surveillance programs. You would like to know how to be heard, for your ideas are good but nobody seems to pay attention to you.

You would like to know why it always seems to be better to change jobs than to change your current job.

Frustrating, huh? The truth is, I have no clue. I am a software developer myself, and I have long suffered this mismatch. I am not going to offer solutions or a checklist for people to follow and to try to surf on top of conflict situations.

In a previous article of mine provides a very simple tool that can be used in this respect. I actually mention, at the end of that text, that the aim of saying “No” is, ultimately, the establishment of a dialog. The importance of this fact should not be underestimated; it is not just saying “no” for the sake of it, but to generate a dialog. Should this dialog not happen, then yes, I suggest learning more about politics, and in the worst of cases, just leaving for other horizons.

I can totally understand and relate to the struggle generated by teams that cannot communicate within themselves, let alone with their surrounding communities. I do think however, that the lack of communication is not the fault of either party separately, but rather, of both.

In other words, if the dialog I mentioned above does not happen, if a communication channel does not work, the problem lies at both ends at the same time. We are also responsible for our broken teams, just like all other team members are.

Making iOS Applications Accessible

Adrian Kosmaczewski

2016-09-15

This is the talk I gave at the Mobile Developer Summit, Bangalore, India, September 15th, 2016

Thanks everyone for attending my talk, and thanks for the organizers of the Mobile Developer Summit to invite me during all of these years since 2011. I'm very glad to be here.

I will start this talk with a story.

In 2010 I was part of an itinerating group of people trying to get developers to pay attention to the iPhone and the recently released iPad. We gave talks about the subject in Denmark, Zurich, Geneva and finally in London, more or less during the time of the football World Cup in South Africa – I know, not the best moment for a developer conference, but still, that's how things work sometimes.

In that conference I gave a talk that I also gave here in MODS 2011, the one called "Ten Commandments for iOS Development;" maybe some of you might remember from my last time here.

I gave my talk, and at the end, as I was wrapping my cables and packing my laptop, a man approached me and told me, "you forgot one commandment." Still distracted by the jungle of cables in my bag I did not immediately raise my head, so I asked him directly as I was still storing items.

"Really? What is the commandment that I forgot?" I asked.

"That of making apps accessible."

I raised my eyes and I saw a tall blind man next to me, with his cane and an iPhone in his hand. I stopped packing my items and I got up, really curious.

The man, whose name I have unfortunately forgotten since then, told me his story. He was a software developer himself; he worked in London in an agency, and, understandably enough, specialized in making apps for users with viewing difficulties and disabilities.

He proceeded them to quickly demo his app to me, and to my absolute amazement he was able to use his iPhone without any kind of assistance, just by using a feature called VoiceOver, which I will explain in detail later in this talk.

Needless to say, I was ecstatic. Talking to this man was one of the most enlightening moments in my life, a real eye-opener, and since that day I have tried to bring the topic of accessibility to all of my fellow developers, in every project that I have worked on since then.

Disabilities

The World Health Organization defines disabilities as follows:

> Disabilities is an umbrella term, covering impairments, activity limitations, and participation restrictions. An impairment is a problem in body function or structure; an activity limitation is a difficulty encountered by an individual in executing a task or action; while a participation restriction is a problem experienced by an individual in involvement in life situations.

There are three major kinds of disabilities:

- Impairments
- Activity limitations
- Participation restrictions

It turns out that 1 billion people in the world have a disability of some kind, of which around 300 million people are either blind or have some kind of vision impairment. 20% of Americans, for example, have a disability of some kind. One out of every 200 women and one out of every 12 men has color blindness.

Amazing, right? It turns out that by ignoring the problem, we make it worse.

A very important concept to keep in mind that disabilities are not a barrier: the barriers are something created by society, when we deny basic rights of movement, use, and enjoyment to some fellow humans. It is our duty to change our mindsets, and to make our applications as accessible as possible. We must not make assumptions about the capabilities of the users of our applications!

Actually, it turns out that this is a legal duty.

Convention on the Rights of Persons with Disabilities

The Convention on the Rights of Persons with Disabilities is an international human rights treaty of the United Nations intended to protect the rights and dignity of persons with disabilities. Parties to the Convention are required to promote, protect, and ensure the full enjoyment of human rights by persons with disabilities and ensure that they enjoy full equality under the law.

Article 1 says that the purpose of the convention is

> to promote, protect and ensure the full and equal enjoyment of all human rights and fundamental freedoms by all persons with disabilities, and to promote respect for their inherent dignity

Guiding Principles

There are eight guiding principles that underlie the Convention:

1. Respect for inherent dignity, individual autonomy including the freedom to make one's own choices, and independence of persons.
2. Non-discrimination.
3. Full and effective participation and inclusion in society.
4. Respect for difference and acceptance of persons with disabilities as part of human diversity and humanity.
5. Equality of opportunity.
6. Accessibility.
7. Equality between men and women.
8. Respect for the evolving capacities of children with disabilities and respect for the right of children with disabilities to preserve their identities.

The Convention stresses that persons with disabilities should be able to live independently and participate fully in all aspects of life. To this end, States Parties should take appropriate measures to ensure that persons with disabilities have access, to the physical environment, to transportation, to information and communications technology, and to other facilities and services open or provided to the public.

What is Accessibility?

What is accessibility, then?

Accessibility is the degree to which a product, device, service, or environment is available to as many people as possible.

Accessibility is not to be confused with usability, which is the extent to which a product (such as a device, service, or environment) can be used by specified users to achieve specified goals with effectiveness, efficiency and satisfaction in a specified context of use.

Accessibility is not only about disabled people, and it is not only about apps or websites. Accessibility is for everyone. Accessibility is about everything and everyone, to ensure a fair degree of access to goods and services and places and situations, to as many people as possible, at all times.

For example, you can ensure that your conference is accessible by making it friendly to people with disabilities, but also by making it friendly to people coming from far away, to people speaking other lan-

guages, to people from other cultures or from any other social group you might think of.

You can also design your cities to be accessible, of course, but there are still far too many places in the world where walkways, public transportation, buildings and homes are just not accessible enough, like New York City, for example.

But most of you are developers, and as such the first place where you can change the world is through your applications. And your applications are born in your teams, and Apple recognized the fact that including people with disabilities in their teams made their teams produce better apps.

Not only that, but you can even shop accessibility accessories in the Apple Store as well; this is an active message to a large community, telling them that they care. Of course this is a great business, as well, why denying it? The opportunities are endless in this space, and the time is right for us to step in.

All of this is happening, right now, and this talk will be a call to action, to all of you, to become better software developers through accessibility.

Making Accessible Apps

As I told you at the beginning of this session, the accessibility features of macOS, iOS, watchOS and tvOS are unparalleled and extremely powerful. Apple has historically had a strong commitment to inclusiveness, and many features are available to developers to increase the reach of their applications:

- In all the operating systems:
  - Internationalization
  - VoiceOver
  - Color adjustments (Grayscale, etc)
  - Dynamic Type
- In iOS:
  - Magnifier
  - Typing feedback
- In watchOS:
  - Taptic time
- In tvOS:
  - Switch control

Applications can provide accessibility:

- Motor: for example, the dwell control in macOS.
- Vision: for example, the new iOS 10 magnifier, or VoiceOver.
- Hearing: for example, visual cues instead of sounds.
- Learning: for example, for users with autism and dyslexia.

When designing and developing a new application, accessibility has to be considered seriously from the start. I am going to talk about the different aspects to take into account in the different phases of the creation of an application, that is, during design, development and testing.

We must engage with people with disabilities during these phases, in order to learn more about their needs, and to modify our designs and our code in order to make the final result accessible and easier to use.

Design

When designing an application with accessibility in mind, there are five areas to consider:

- User Experience
- Typography
- Layouts
- Color
- Iconography

Let's talk about each of these in detail.

User Experience At the very beginning of a project, when the idea of an application starts to take shape, the design team should consider the experience of users with disabilities as a requirement. This includes creating the required user personas and scenarios for them, the required mockups and their corresponding user testing sessions, so that the needs of all types of users is considered thoroughly.

In some countries, particularly for government projects, there are written guidelines regarding accessibility, and in many cases they are enforced by authorities. Pay attention to those guidelines, as well as those provided by Apple for each of its platforms, so that you begin your projects by taking the right decisions. In those cases you not only might risk a rejection from the App Store or a bad experience for your end users, but flatly the rejection of the final product by the government authorities, and you do not want that to happen.

Typography Typography is a very important element in application design. Through hierarchy, designers are able to highlight elements and guide through the various steps required to perform a task.

There are several techniques to create hierarchies in text:

- Spaces
- Size
- Bold
- Italic
- Color
- Upper- and lowercase

Whenever you design an application, keep in mind the final languages in which this application should be available. This information is very important, because not all writing systems support uppercase or italic. In general, the best strategy to manage hierarchies is to use larger font sizes.

Needless to say, some languages like German have very long words, and they take more space on the screen.

Also, please stick to standard fonts whenever possible, or at least make sure that the custom fonts you use support bold styles, to use those as a basis for your hierarchies.

Finally, and here's a tip for both designers and developers: the `clipsToBounds` property should be disabled in Interface Builder, because some alphabets have diacritics below the descender or above the ascender – this means that using bigger line heights is a must in your designs.

Layouts Dynamic layouts are very, very important, for the simple reason that there is an accessibility feature called "Dynamic Text." With this feature, that all application should be supporting, users can request the text of all applications to be bigger – and sometimes very big! – or smaller than the standard size.

This means that your designers should not only give you the font specifications for the application, but rather a table where the following styles are available in all the following heights:

Styles:

- Title 1
- Title 2
- Title 3
- Headline
- Body
- Callout
- Subhead
- Footnote
- Caption1
- Caption2

Sizes:

- xSmall
- Small
- Medium
- Large (Default)
- xLarge
- xxLarge
- xxxLarge

Using this table, you should be able to test your application in all the sizes, and this will be a great help for the end users of your application.

Pay attention to the possibility for users to select the “Bold Text” option for text in all operating systems, to increase the legibility of text.

Finally, remember the rules for right-to-left languages; there are two of them, Hebrew and Arabic, and when your application displays those languages, all the UI should be reversed. However, pay attention to the fact that the following elements must appear in left-to-right mode, even in the case of Hebrew and Arabic:

- Numerals
- Phone numbers
- Clocks
- Music notes
- Graphs
- Video playback controls
- Images

Color Color is a great tool for conveying emotions to the user. The most common choices can be quickly summarized as follows:

- Blue is the most popular color, it has a short wavelength and represents peace, quietness and loneliness.
- Green represents nature, good luck (in Western cultures) and safety.
- Red has a long wavelength, and usually conveys the ideas of revolution, conflict, passion and love. It also represents good luck in asian cultures, which is shown in the Stocks application for example.

There is a lot of people with color blindness; one out of every twelve men, and one out of 200 women; that is, around 700 million humans have this condition!

For your designers, then, the most important thing to keep in mind is to always work in their designs using black and white screens! This is a very simple trick but it always work.

How do you create contrast with grayscale? Well, keep in mind that the ratio of contrast between black and white is 21:1, while between gray and black is only 4.4:1 and only 1.9:1 between dark gray and black. This means that you should not use dark gray for disabled elements! Use it only for decoration. For disabled elements, use a lighter shade of gray, or a color that translates as easily into that shade.

It is very important that your designs always have higher contrast ratios, so that they work good in grayscale environments. The contrast between the elements on the UI should not rely only on color alone. Use geometry, contrast, typography for that.

Remember when you are designing your user interfaces, to work on grayscale screens. You can change that setting very easily in macOS, iOS, tvOS and watchOS. If your UI works well in grayscale, then it will

work even better with color. And, if possible, test the application with users with viewing disabilities, to make sure that your designs are usable by everyone.

Iconography Iconography is as much an art as a science, most of which is derived from Semiotics, that is, the science of meaning. Given the differences in the cultures of this planet, you should not assume a given meaning for an icon, as different people will interpret them in different ways. Cultural, religious, technical and moral reasons might cause your users to feel offended by the iconography of your application.

So, here's a simple tip: when in doubt, test with real users, or if all else fails, just use text. Speak to your users and tell them your story! Sometimes a good word might be all you need.

If your UIs have to work on Arabic or Hebrew, use non-directional icons that work in both left-to-right and right-to-left orientations.

Simple guidelines: First and foremost, read the Apple Guidelines for the platforms you are working on; they contain a wealth of information regarding accessibility.

Here go some other simple tips and tricks to design accessible applications:

1. Add descriptive text in UI controls
    - Why? Users cannot see everything, devices can "read" aloud
        - For whom? For blind users and when assistive devices cannot help
        - How? ALT attribute, android:contentDescription or accessibilityLabel + accessibilityTrait in iOS
2. Minimize text input in UIs
    - Why? To avoid mistakes and frustration
    - For whom? Everyone, actually; particularly for motion disabilities, also for shaky situations (bus, train)
    - How? Avoid free text; choose from lists; default entry modes for different types of text; keyboard shortcuts if available.
3. Avoid scrolling
    - Why? Small screens
    - For whom? Users with restricted vision or using screen zooming devices
    - How? Clear language; limit scrolling to one axis (usually vertical); avoid large images; position important elements on top and visible at first; large clickable areas.
4. Do not use gestures alone; complement with buttons or links
    - Why? Not intuitive, unknown to most users
    - For whom? Blind users, temporary disabilities (plaster), users in the street.
    - How? Add links and buttons; add text to explain.

5. Allow zooming of UI elements
    - Why? It can be blocked, particularly in web pages
    - For whom? Users with bad sight
    - How? HTML5 meta tags controlling the viewport
6. Add captioning to video content
    - Why? Because this can be read by screen readers for blind users, but it is also useful if for some reason the user cannot increase the volume of a video, and wants to know its contents.
    - For whom? For everyone, but mainly for users with hearing problems.
    - How? Subtitle files and adding chapter information in long video files.

Development

Developers can increase the accessibility of their applications using different mechanisms. I am going to explain them in this section.

First of all, you should get acquainted with the `UIAccessibility` protocol (and, if you are creating a macOS app, you should check out the analog `NSAccessibility` protocol instead.)

Just like any protocol in Cocoa, and as the word "protocol" suggests, the `UIAccessibility` protocol represents the phases of a conversation between the accessibility subsystem of the operating system and your user interface. Each of the methods in `UIAccessibility` represents a question to which your UI will give an appropriate answer in time.

The main four questions that are asked through `UIAccessibility` are the following:

1. Are you accessible?
2. If yes, what are you?
3. And who are you?
4. And where are you?

The first question is trivial but fundamental, and it is answered by the `isAccessibilityElement` boolean property. If you answer `true` to this question, then you should provide suitable answers for the following three questions.

The second question is answered by the `accessibilityTraits:` property, which holds a value of the `UIAccessibilityTraits` enumeration. This enumeration is a bitmask, which means that any `UIView` can be many of these.

The answer to the third question is provided by the `accessibilityLabel` property, which returns a string.

Finally, the answer to the fourth question is given by the `accessibilityFrame` property, itself returning, predictably enough, a `CGRect`

value.

If your application uses only standard iOS `UIView` subclasses, then you do not need to do any of this; all of the standard iOS components already provide this information for your automatically. But of course, if you create your own `UIView` or `UIControl` subclasses, then you must provide the information required by the `UIAccessibility` protocol in your code.

Developers can also use Interface Builder to provide the information required by the `UIAccessibility` protocol, instead of using code. The "Accessibility" panel in the Identity Inspector at the right of Xcode provides a UI that allows you to provide the required information by the accessibility subsystem of your target operating system.

And finally, we come to the possibilities of internationalization of your applications. First, using the `NSLocalizedString()` family of functions, developers can separate localizable strings from their application into individual `.strings` files, usually saved with the UTF-16 encoding, and which can be handled to translators so that your application can be used by users in different cultures and languages.

The second important feature of internationalization is the possibility to localize storyboards and XIBs, which is particularly important when supporting right-to-left languages such as Hebrew or Arabic. In those languages, your UI should be reversed as well, so localizing your UI files helps you achieve that.

I am not going to say much more about internationalization, because Laura Savino is going to talk about that tomorrow at 10:50 in the SD Hall, so please make sure to attend her talk called "iOS Localization with Modern Xcode Tools."

Testing

To test applications for their accessibility, previously one had to manually check applications (code and Interface Builder files) or to have people manually testing the applications, to make sure that the user experience matches the expectations.

There are good news from Xcode 8, however; the brand new version of Xcode for iOS 10 and Swift 3 brings a new tool with it: the Accessibility Inspector, available as usual in the Xcode menu, under the Tools submenu. Using this inspector, developers can get a complete report of the current state of the accessibility features of applications, and using this information, they can add the missing parts.

As complete and as awesome as the Accessibility Inspector is, I cannot stress enough the importance of having actual users, preferably with disabilities, testing the applications and providing feedback to the design and development teams.

There is, however, one major advantage to adding accessibility information to applications; and that is, Xcode UI Testing. It turns out that the UI testing capabilities of Xcode make use of the accessibility features of applications, in order to find and interact with UI controls. By adding accessibility information to your iOS applications, you are actually enabling a whole new range of automated testing, inside of Xcode itself.

Demo

Before starting the demo:

- Prepare Mouseposé
- Open a QuickTime document recording the screen of the iPhone connected by cable, so that one does not have to unplug the computer.
- Use the iPhone 6S simulator for the app demo

Introduction In this demo I am going to give you a small, very small overview of the different accessibility features available to iOS developers, many of which are very similar to those available in the other operating systems, like macOS, tvOS and watchOS.

Overview of iOS Accessibility Features First of all, let's dive into iOS and let's take a look at the possibilities. The Settings application allows users to enable and disable a plethora of options about accessibility; let's take a look at some of them.

First and foremost, VoiceOver, which is a technology available since iPhone OS 3 and the iPhone 3GS, which allows users with vision impairments to have the operating system "read" whatever is on the screen. We are going to use this feature in the demo.

Another interesting accessibility feature in iOS 10 is the AssistiveTouch control, which provides on-screen functionality that is otherwise available through gestures. This helps users with mobility or learning disabilities to use iOS more effectively. It is also a great feature if your hardware buttons are broken; I used this feature extensively with an old iPhone 4S I had a couple of years ago, whose wake button was broken.

iOS 10 also brings a new accessibility feature, the Magnifier; this feature was mentioned during the last WWDC keynote, and it is very simple to use. Just enable the feature and then triple-tap on the home button to launch it.

The App But let us go to the heart of this demo.

I have created a small iOS application that loads the contents of a JSON file contained within the bundle of the application, and displays the data on a `UITableView`. The application in that sense could not be

simpler. As you can see in the storyboard, it is a classic master/detail application, only for iPhone.

By the way, if you see me zooming in and out during this demo, this is because I have enabled yet another accessibility feature on macOS: Go to the System Preferences, Accessibility, Zoom and select “Use scroll gesture with modifier keys to zoom” and then I just press Control and zoom with a gesture on my trackpad. Very useful for teaching!

Let me launch the application so I can show you what it does. First it shows the data on a standard table view, and when I tap on any of these entries, a small form appears for the user to modify the information. The form features a very rudimentary form of validation; if any of the fields is empty, the name of the field appears in red to indicate that it should not be empty.

Nothing else but enough for this demo.

All of the components in this application are standard, except for the small “favourite” star control, which is an extremely simple subclass of `UIControl`.

VoiceOver in Detail I will now activate VoiceOver, and will try to navigate back to the application just by using the voice feedback, and without looking at the screen. As you can guess I have rehearsed this a few times, but it is still challenging. We take for granted our sight so much.

As you can see, VoiceOver does quite a good job highlighting the currently selected item in the application, and reading back aloud whatever its value may be. But still, in the application itself the feedback could be better. In particular, when I select an item on the list by double-tapping, and then I dive on the form, the favourite custom control says “star” and “white star” which are the names of the Unicode characters that I have used in my control to show both states. Not very helpful, is it?

Also, you can imagine that developing a complex application and checking it through this procedure might become quite tedious, so again we can do better than that.

Accessibility Inspector Enter the Accessibility Inspector, a new feature of Xcode 8, to increase the accessibility of an application. We can access it by selecting the “Xcode / Open Developer Tool / Accessibility Inspector” menu.

The Accessibility Inspector is a very simple application that has three main screens:

- The Inspection screen
- The Audit screen
- The Settings screen

The toolbar also features a menu, to inspect any application running either in the current Mac or in any connected iOS device, and a selector, allowing you to point to a specific control or view on the screen and find out its accessibility settings.

The Inspection screen shows the state of the currently selected item, exactly as the accessibility subsystem can see it and interact with it.

The Audit screen performs a quick scan of your application, and gives you a report about the major problems found with the application.

Finally, the Settings screen allows you to quickly set your app in inverse colour mode or to change the Dynamic Type settings, which triggers the required `NSNotifications` as expected. Speaking about which, you can open a separate window in the "Window / Show Notifications" where all the notifications sent are displayed.

Auditing the App So, let us run an audit on the master screen, and very quickly we see that there are absolutely no warnings. This is because most standard UIKit controls already provide accessibility information by default.

However good this is, we can make it better. Remember when I hovered my finger over the individual cells? VoiceOver would read to me the whole contents of the cell, which is a bit ridiculous, because just the name would be better in this context. For that we are going to edit the code of the application a little bit, and we are going to specify a default value for the `cell.accessibilityLabel` property.

I am going to go now to the settings panel to check if Dynamic Type works well; as you can see we can change the sizes of the text, but unfortunately the height of the cell stays the same. By making some small changes in the `viewDidLoad` method of the master controller, we can tell it to automatically resize those objects as well.

Let us try VoiceOver now; yes, now the system only says the first name of the customer, and it sounds much better and useful.

The Form Screen Let us pay attention now to the detail / form screen. First of all, a simple usability fix that actually helps everyone is to use the "numeric" keyboard for the "age" field; this will benefit the application greatly, because it minimises the effort required to enter data. Accessibility is about everyone, remember.

In the Accessibility Inspector (whose window can stay permanently on top of the others, as you can see) I run an audit of the form screen and I can see that there are a few problems related to Dynamic Type. Let us solve these problems in the code.

First of all, to react to Dynamic Type changes, we have to listen for the `UIContentSizeCategoryDidChange` notification, and in the notification handler we call the `UIFont.preferredFont(forTextStyle:)` method, passing the name of the desired style as a parameter.

Now, having done that, we can see that the application reacts correctly to the font changes. However, pay attention to the fact that instances of the `UITextField` class do not change their height; you might want to use other components such as a `UITextView` if required.

You can also see how AutoLayout handles automatically the resizing of your controls, and how the fields keep their baseline aligned to that of the labels next to them.

Validation errors are another interesting problem; if I set my Mac to display in grayscale, you can see that the contrast of the error message is not great; so what we're going to do is to use an error label at the bottom of the form, and we are going to display it. Also, the labels of the fields that do not pass validation will display an asterisk, to make them more prominent and visible. This way we avoid using colour as the unique way to convey information, and we increase the contrast.

I am also going to add a vibration effect, to make it even more clear to all users that the application has a problem. But the truth is that blind users would not know exactly what is happening. We are going to add now a text-to-speech help voice that will explain the problem to the user, but this has to be done only if VoiceOver is activated.

Thankfully, the `UIAccessibilityVoiceOverStatusChanged` notification and the `UIAccessibilityIsVoiceOverRunning()` function tell us exactly that. So, now, if VoiceOver is running, when there is an error, we are going to ask the iOS speech synthesiser subsystem to read the error message out loud.

Another important change consists in enabling proper accessibility to our "favourite" control. We can do that directly in the source code of the control, by answering the three questions that the `UIAccessibility` protocol will ask it every time it is required.

Of course this can also be done on an instance-basis, instead of directly at a class basis; you can do this in code, or in Interface Builder. Actually, we are going to use now Interface Builder to add the required information for the text fields, which by themselves are not very useful to VoiceOver. For that, we are going to use the "Accessibility Value" field on the Interface Builder inspector of our fields.

Finally Running VoiceOver on our application now yields much better results. The cells on the table only say as much as needed; the screen also reacts to Dynamic Type properly. The form is much better, too: it has better contrast for colour blind users, provides better information to VoiceOver users, shows a better keyboard to enter numeric values, reacts properly to Dynamic Type and it even speaks out the error message if needed.

The Accessibility warning now displays no warnings at all, and our testing shows that the user experience is good.

Not only that, but our application is now ready for UI Testing in Xcode, because, precisely, those kind of tests use accessibility information to access the different items on the screen. In this case our test selects the first item in the list, and then taps the favourite control to test its value and behaviour.

Conclusion

Accessibility is a fancy word for empathy.

Making our apps accessible is the way of the app maker to make the world a better place. It is a way to apply our religious sensibilities, whatever your beliefs may be, into helping the other. Going to the other. Projecting yourself with empathy, not by saying "I am sorry for your situation," whatever that may be, but actively trying to understand the other, and offering ways to overcome those situations.

Because accessibility is empathy, it is also internationalisation. It is all about telling the other "I want you to understand me," which of course requires me understanding you.

Because accessibility is empathy, it is also inclusion. Understanding other ethnographic groups, other religions, other nations, other cultures, other sexual orientations makes us better people. Including the other in the development team will invariably yield better applications, and yes, if that is what you want to know, that also translates in more sales and more money.

So, in the end, through accessibility we all win; both materially and non-materially speaking. The "side effects" of accessibility, those which economists often talk about in their papers, are tremendous. The overall balance for society is incredibly positive.

And in my humble opinion, India is in a particularly good position to lead this change, because of your diversity. I see the richness of your country, I see so many people fluent in lots of different languages, I see your culture, your music, your food, your natural kindness, and I see a country that is naturally biased to inclusiveness and accessibility.

I leave you with a quote by Helen Keller, writer, political activist, and the first deaf-blind person to earn a bachelor degree:

> Science may have found a cure for most evils; but it has found no remedy for the worst of them all - the apathy of human beings. Helen Keller, My Religion, 1927.

We as developers can embrace empathy, and fight apathy, through accessibility. Please make sure that your apps are accessible. This is very important.

My name is Adrian Kosmaczewski, and this was Making iOS Applications Accessible.

Thank you so much.

References

The following WWDC sessions have helped me to create this presentation and I strongly recommend you watch them too:

- WWDC 2016:
  - 104: Disability & Innovation: The Universal Benefits of Accessible Design
  - 201: Internationalization Best Practices
  - 202: What's New in Accessibility
  - 232: What's New in International User Interfaces
  - 407: Auditing Your Apps for Accessibility
  - 801: Inclusive App Design
- WWDC 2015:
  - 406: UI Testing in Xcode

Courage

Adrian Kosmaczewski

2017-01-31

It is 7:30 AM.

I grab my iPhone and I place my thumb on the home button. My iPhone pretends that my finger is not my finger is not my finger. I manually type in my PIN code instead. I put on my WATCH. It does not unlock itself automatically, even if I follow the ritual dance of holding my WATCH up while I unlock my phone with my thumb. I end up manually entering my PIN on my watch, too. I go to the kitchen to make a cup of coffee.

I go to my desk. I open the lid of my MacBook with Touch Bar. I have my iPhone in my pocket and an unlocked WATCH on my wrist. The Mac should unlock itself automatically. It does not. I place my finger on the Touch ID scanner. Wrong finger. Good finger. Not recognized. I end up manually typing in my password.

I start iTunes. Just like every Monday, it tells me that I need to log in to Apple Music. I do not remember ever having logged off. I enter my username and password. The dialog goes away. iTunes still does not allow me to listen to music. I close and re-open iTunes. I log off and on a few times. I finally reboot my Mac. I discover that the artist I would love to listen to that morning is not available on iTunes Music. I select another artist. I hit play. Music does not come out from the built-in speakers. I plug in my old 2002 Harman Kardon SoundSticks. The music plays.

I select the AirPlay speakers connected to my Airport Express to enjoy some music while I have breakfast. The Mac cannot connect to them. I reboot the Airport Express. After a few minutes I can stream music. After 2 minutes the stream ends. No way to start it again. My coffee is cold.

I sit on my Mac and open a Pages file stored on iCloud, one I was working on my iPad Pro during the weekend. The sync fails and I cannot see the last modifications I made on my iPad Pro. Open and close apps on both devices. I reboot them both. Pages for Mac tells me that there is a conflict between the versions in both devices, even though I have never edited the file on the Mac. I select the version on the iPad. My changes are lost.

My WATCH vibrates. Wife sends me an iMessage. Although she is

on my contact list, the WATCH only displays her phone number. I try to reply directly from the watch but I hit the wrong button on the tiny UI and send nonsense. I try to cancel the operation but the “dismiss” button is out of sight. I hit the crown and the side button. The watch keeps nagging me with vibrations for pretty much every single notification that I get that morning. I take it off and leave it on the desk.

I open iMessage on my Mac and the messages of my wife are not there yet. I write my reply to her anyway and hit enter. I am surprised my message shows up as “delivered.”

I open Xcode. I have to debug some iOS code. I plug my iPhone to my Mac. Both my Mac and my iPhone pretend they don’t know each other. Yes, you can trust each other; I told you so every day for the past 5 months. iPhoto opens thinking that I want to sync pictures. iTunes opens and starts downloading updates. I cancel the downloads. I close iTunes. I close iPhoto. Both take a few minutes to close. They eventually close.

By that time I have totally forgotten the bug I wanted to fix on my iPhone app. Ah, yes, I remember now. In the meantime the messages of my wife arrive to my Mac and plenty of notifications pop up on the screen, including things we wrote the previous week.

I open my Xcode project. The certificates, of course. Hit the fix button. I’m surprised it works. I need more coffee. The long compilation times of Swift 3 are perfect for a pause. At least the compiler does not crash.

I come back with fresh coffee. I sit down to write some code. Xcode opens the autocompletion menu at every keystroke. I disable it. I sigh and close Xcode. I open AppCode.

The Touch Bar suddenly stops showing the icons for the volume and the music. I do not know why. The “Esc” key is the only one showing up, which is quite ironic when I think about it. I touch the area where the volume control is supposed to be and everything appears again. I remember when the emoji bar appeared on the middle of the screen the first time I wanted to use it.

I would love to know how long I have until my battery dies, but the time indicator suspiciously disappeared a couple of weeks ago.

Debugging with AppCode requires hitting F7 and F8. I basically turn on the feature that keeps the “F” keys always on in the Touch Bar. And because of the lack of feeling on my fingers, I end up hitting a different key every time that my eyes are fixed on the screen debugging. The keys disappear after a few seconds. They reappear as soon as I touch the Touch Bar.

I sigh. I plug my USB keyboard to the Mac. I need a USB to USB-C dongle. I plug everything in place. The keyboard does not work. I unplug and replug a few times. The keyboard works. I wonder for how long. In the meantime the batteries of my Bluetooth Apple Trackpad

die. I change the batteries. My Mac does not recognise the Trackpad anymore. I reboot my Mac. The Trackpad is there again.

I open another application on my Dock. I am told that it cannot be opened. I click again. No luck. I remember that I got it from the App Store. I remember the certificates issue. I delete and re-download it from the App Store. The app opens. I forget why I wanted to open it in the first place.

I try to open an application I bought yesterday on this Mac. The operating system protest, telling me that I have to login to the App Store because the application was bought in another Mac. I log in. The app opens.

I need to print the Pages document I was working on. I turn the printer on, ask Pages to print it. The Mac complains of a CUPS, JetDirect or Ghostscript issue. My coffee is cold again. I reboot the Mac and the printer. I realize there is no paper on the tray. The page prints a blank page. The printer needs ink.

The iTunes icon starts jumping up and down in the Dock. I click on it. It asks to log in to the App Store. I don't know why. I click cancel. I quit iTunes.

I start Xcode. It crashes and quits. The dialog says "Unexpectedly."

I shut down Mac, iPad, iPhone, WATCH and Airport Express. I shut down my TV too. Just in case.

It is 9 AM.

Android for iOS Developers

Adrian Kosmaczewski

2017-03-24

Just published my new book: “Android for iOS Developers”.

The world of mobile development is a ground in constant motion. However, for the past five years, Android and iOS have both reached the level of dominant players in the field, moving other platforms out of sight. Due to the complexity of these systems, developers tend to concentrate their efforts in just one platform; however businesses must target both platforms to remain competitive in the mobile market.

This book provides an iOS developer’s perspective on Android, highlighting the similarities and the major differences between both platforms. The author hopes that these lines will help other developers to jump to the fascinating world of Android using their hard earned iOS knowledge.

This book is intended as a step-by-step guide to guide developers well versed in the arts of iOS into the realm of Android mobile application development.

Read it here:

- PDF[1]
- HTML[2]
- EPUB[3]

[1] https://akos.ma/books/Android_for_iOS_Devs/Android_for_iOS_Devs_1st_Ed_2017.pdf

[2] https://akos.ma/books/Android_for_iOS_Devs/Android_for_iOS_Devs_1st_Ed_2017.html

[3] https://akos.ma/books/Android_for_iOS_Devs/Android_for_iOS_Devs_1st_Ed_2017.epub

The Developer Guide to Migrate Across Galaxies

Adrian Kosmaczewski

2017-04-28

This is the presentation I gave at the second App Builders Conference[1] in Lausanne, Switzerland, April 25th, 2017.

Last year I briefly touched the concept of “technological galaxies” in my bestseller talk of all time[2], the one I gave in Zurich and which you have most probably recommended or shared on social media.

Figure 1: A galaxy

So, first of all, thanks a lot for that.

My current interests around computers are very far away from the usual subjects that you can find in events such as this one; I am not the slightest interested in knowing if reactive programming is the way for the future, or if you should just use AsyncTask or dispatch queues

[1] https://appbuilders.ch

[2] /blog/being-a-developer-after-40/

or futures or promises or just function pointers and unleash all hell through cascading callbacks.

I just do not care much about that anymore.

I firmly believe that the biggest issues we have in the industry are social problems. And I share this diagnostic with other developers around my age—yes, I reached the point in my life where I can openly say "people my age;" that is a bonus of getting old, that and the odd realization that I am halfway in my professional life.

So let us talk about six galaxies, shall we? Just six. In just half an hour. Believe me. Everything is connected.

1. Perl, Python, Ruby, and PHP (aka Babelia)

Figure 2: Babelia

All vendors want to keep users and developers in their platform at all costs. To reach that goal, they will do whatever they can to make their both groups happy, by constantly improving their systems for better security, usability and enjoyment—NOT.

This is not what happens. No, not at all. Quite the opposite. **The truth is that all software rots.**

And it rots continuously, unstoppably, a juggernaut of spaghetti, a meteoric ball of mud[3] hovering upon our world. But is this not absolutely counterintuitive? Is software not a physical thing after all, let alone an organic thing? How could it rot? One can spot the occasional website

[3] http://laputan.org/mud/

of an independent software vendor claiming that they are “artisanal” and “organic.” This is typically what hipsters do.

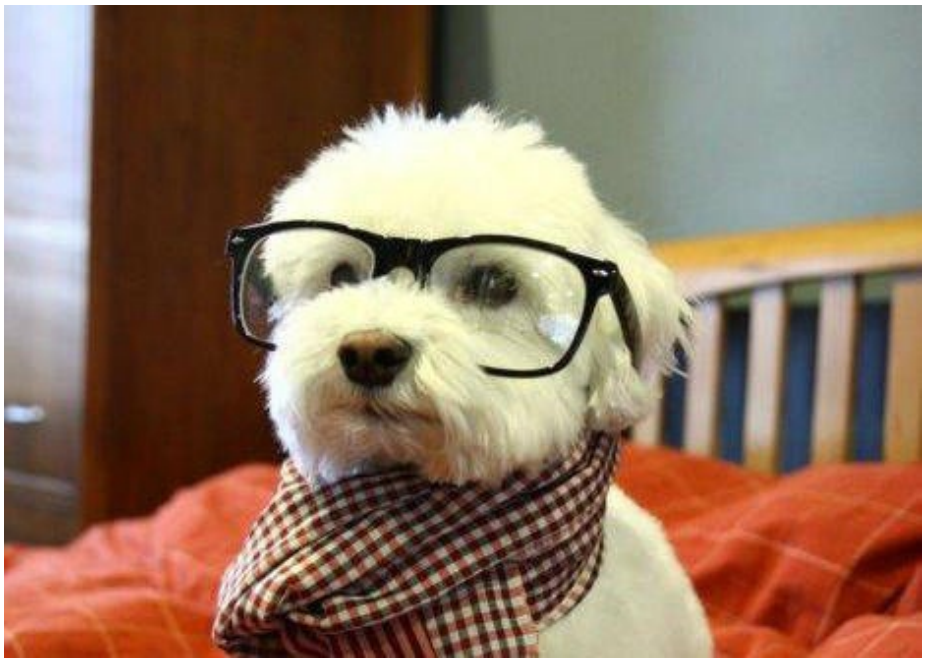


Figure 3: Canis Hipsterius

Ahh... hipsters. They speak Japanese and have Italian espresso machines and they bike to work in the middle of a big city breathing the unmistakable smog of large urban centers, and then they go to holidays in Djibouti and have dogs of some weird race and they have great taste and they make beautiful software that nobody buys.

Because our industry is broken by design. We spend days, months, years building apps, and at the end it means nothing because you are not Marco Arment[4], and you are not friends with John Gruber[5], so it means that nobody knows you exist. With a bit of luck you will be able to get some money, maybe enough for a brand new set of dongles for the next MacBook. Maybe.

And then you attend these conferences where everybody is talking about massive view controllers and reactive and optional and enumeration types and you realize with guilt and dispair that you are still writing your code as if it was 1985 and Perestroika[6] had not yet happened and you still had your Walkman[7] plugged to your ears listening to New Order[8].

[4] https://marco.org/
[5] https://daringfireball.net/
[6] https://en.wikipedia.org/wiki/Perestroika
[7] https://en.wikipedia.org/wiki/Walkman
[8] https://en.wikipedia.org/wiki/New_Order_%28band%29

Figure 4: Dongle Galore

We are all faking it. Let us first agree on this.

But I digress. I said I hate software vendors. So the first thing you have to do when you work with computers is to make sure that your files are as cross-platform as possible. Which at the beginning it is a bit of a problem, because when you do not know that there are other types of computers, you just use Windows 3.1 and then you save all your files in the 1992 version of the Word document format using the Windows-1252 encoding.

True story. Because you just do not know any better, that is all. Even worse, I had the bad idea of using the Equation Editor of this Word thing to spice up my university papers. I did not know that LaTeX existed!

And then all of a sudden it is 2017 and you realize that you have megabytes and megabytes of equations stored in files that only LibreOffice[9] or AbiWord[10] can open; well, almost.

Remember Pages? Well it turns out that Pages has also an equation editor, but unlike Word it accepts LaTeX—and even MathML… but there was another problem with Pages until last year, which was the impossibility to open… Apple Pages 2005 documents; a crucial, yet seemingly obvious and astonishingly absent feature.

[9] http://www.libreoffice.org/
[10] https://www.abisource.com/

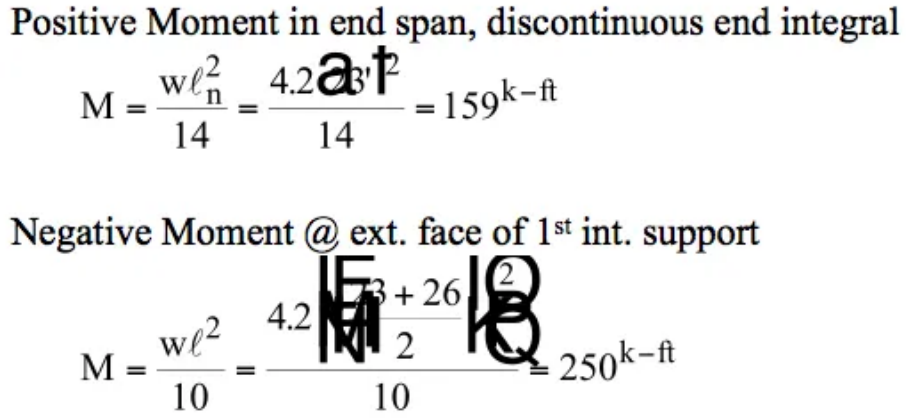


Figure 5: The beauty of the Equation Editor in 2017 opening files from 1992

Let this sink in: Apple Pages 2015 could not open Apple Pages 2005 documents.

In the same backup drive, files from 1992 that I cannot open anymore, together with files from 2005 that I cannot open anymore. There is enough here to hate the computer industry.

Lately I store everything in UTF-8 using Markdown or AsciiDoc formats, including this presentation and its corresponding Medium article, and I use Pandoc to generate anything else that might be needed by someone else. That is right; they are secondary products of a source file that my biographers will be able to open and read in the 22nd century. Because UTF-8 is open and standard and well defined, and because both Markdown and AsciiDoc are done. Good or bad (mostly good) but done. Finito. No changes. Boom.

I do not write documents anymore. I preprocess them. Then I compile them. Sometimes I even link them. Ouuuuuuuh.

I like when some things are fixed, as much as I love when others are not. If it is something where I invest time to learn and write stuff with, I rather enjoy its definition being set in stone, like English, or C++, or Markdown, for example.

As far as the definitions of those things go, I do not think that they will change at all, and in the case of C++, we will still be able to compile C++91 code in the future. Actually, we know for a fact that John Gruber will not change Markdown at all. The only formal specification of Markdown is, and will forever be[11], an implementation in Perl made by Gruber himself, with contributions from the late Aaron Swartz[12] and others.

[11] https://blog.codinghorror.com/standard-markdown-is-now-common-markdown/
[12] http://www.aaronsw.com/

Markdown is what it is, and it will remain like that forever. If you do not like it, well, guess what, people used PHP to create Textile[13]. Somebody used Python to make reStructuredText[14] and AsciiDoc[15]. A guy from Colorado used Ruby to recreate AsciiDoctor[16].

And then a philosophy professor, the king of all hipsters, used Haskell to make Pandoc[17] and brought everything under the same roof. Even Word, LibreOffice, AbiWord and man pages, AsciiDoc and Textile, Markdown and reStructuredText, PDF and RTF, HTML and XHTML, all of them together in just one tool. How about that.

2. VBScript (aka Apocalypsia)

Figure 6: Apocalypsia

[13] https://github.com/textile
[14] http://docutils.sourceforge.net/docs/ref/rst/restructuredtext.html
[15] http://asciidoc.org/
[16] http://asciidoctor.org/
[17] http://pandoc.org/

Those who paid attention to what I said last year in Zurich might remember that I started my career writing code in a wonderful language called VBScript[18].

It had all the bad parts of Visual Basic without any of the good parts, if there were any. There will never be a "VBScript: The Good Parts" book because it would have a negative number of pages.

Visual Basic started its life as a Microsoft product in March 6, 1988, when Alan Cooper showed Bill Gates his system to create GUIs and to add code to those GUIs. Microsoft bought the idea and the rest is history.

Since then, Microsoft integrated Visual Basic into Word and Excel (you can ask Joel Spolsky about that) and also cut it down in little pieces and transformed into this mutant called VBScript.

All clones of Visual Basic shared some common, beautiful features.

On January 3rd, 2010, I provided one of the gazillion answers to the question "Strangest language feature" in Stack Overflow[19]. Of course it is a question without a question mark, because this was still the time when Stack Overflow was a fun place to hang out.

My answer is still there, and it goes as follows:

> In earlier version of Visual Basic, functions without a "Return" statement just "Return None", without any kind of compiler warning (or error).
>
> This lead to the most crazy debugging sessions back when I had to deal with this language on a daily basis.

Jeff Atwood himself left a comment under this answer—for those who might not know him, he is one of the founders of Stack Overflow, and the author of the fantastic "Coding Horror" blog.[20]

> Oh man this sucked. I was so glad they pulled that out in later versions of VB.NET. They really should have just created VB.Sharp and left the compatibility argument on the table.

But this is not all.

VBScript had classes, and classes could have properties: `Property Get` defined getters, while `Property Let` and `Property Set` defined setters. I will let you guess what was the difference between `Property Set` and `Property Let` (hint: it has to do with primitive values vs. object references.)

```
Class Customer
    Private m_CustomerName
```

[18] https://docs.microsoft.com/en-us/previous-versions/windows/internet-explorer/ie-developer/scripting-articles/t0aew7h6(v=vs.84)?redirectedfrom=MSDN

[19] http://stackoverflow.com/a/1995630/133764

[20] https://blog.codinghorror.com/

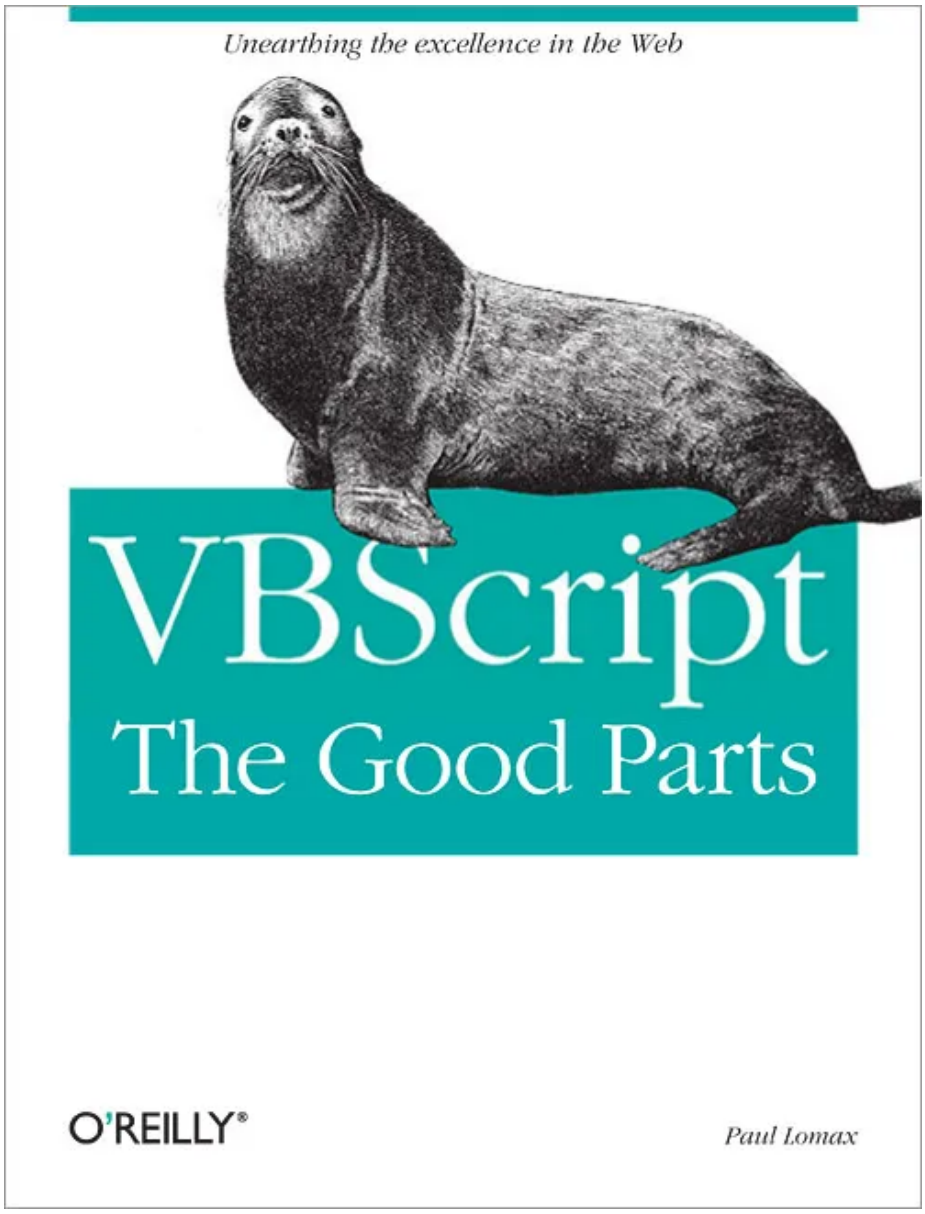


Figure 7: The book that never was

```
    Private Sub Class_Initialize
        m_CustomerName = ""
    End Sub

    ' CustomerName property.
    Public Property Get CustomerName
        CustomerName = m_CustomerName
    End Property

    Public Property Let CustomerName(newValue)
        m_CustomerName = newValue
    End Property
End Class

Dim cust
Set cust = New Customer

cust.CustomerName = "Fabrikam, Inc."

Dim s
s = cust.CustomerName
MsgBox (s)
```

But it turned out you could not store a VBScript class instance in the ASP Session, unless it is a VBScript array or a `Scripting.Dictionary` object. This is because "the ASP server is multithreaded and assigns a different thread to each page request (...). But VBScript class instances (...) must run on the thread that created them."

Even better, you could not inherit VBScript classes: "There is no notion of polymorphism or inheritance in VBScript 5.0. (...) VBScript classes are merely a way to group data and the operations on the data together to improve encapsulation."

In your functions, one did not use the `Return` keyword to return a value from a Function; one had to "set" a variable with the name of the function for that (good luck making a "function rename" refactoring); and if you forgot to return, you would not get a compilation error, not even a runtime error; the function would just return `Nothing`.

```
Function Sum(value1, value2)
    Dim result
    result = value1 + value2
    Sum = result
End Function
```

You could not use parenthesis when calling a Sub with more than 2 parameters. With one, yes, but not with two or more.

All of this is legendary stuff. Let us make a pause and remember the

venerable and still unknown Verity Stob[21], who once said this famous phrase about Visual Basic:

> The four magic constants of the apocalypse: Nothing, Null, Empty, and Error.

Needless to say, the concept of unit tests was non-existent, and error management consisted of `On Error Resume Next`.

```
On Error Resume Next
Err.Raise 6    ' Raise an overflow error.
MsgBox "Error # " & CStr(Err.Number) & " " & Err.Description
Err.Clear    ' Clear the error.
```

And the only way to enforce some kind of sanity was using `Option Explicit`

```
Option Explicit

Dim MyVar
MyVar = 10

' ... and your code explodes (because not declared)!
MyInt = 10
```

Then I became a .NET developer, and all the companies in the area wanted to use VB.NET, instead of C# which was way cooler and coherent as a language. But since .NET was not distributed as part of Windows back in 2004, you had to first install around 20 MB of runtime libraries and frameworks to get your app to work.

Which by itself was not new, because Visual Basic apps, until version 3 at least (which was wildly popular) required you to have the infamous `VBRUN300.DLL` that you were not allowed to distribute for free, and hence there was a market of illegal diskette distribution of copies of that DLL so that people could run your app.

And these days each and every app written with Swift has an overhead of around 8 to 15 MB, and Kotlin apps have one as well, of at least 900 KB, so that they can run on iOS or Android.

In the hierarchy of all things Visual Basic, one can safely argue that

VBScript > Visual Basic > VB.NET

Because, as bad as VBScript was, at least it has been supported continuously in all versions of Windows since NT 4, until today, more than 20 years later.

3. C++ (aka Paradigmia)

Then I went to work for a company that made a financial software package, and a very expensive one for that matter, using C++. I

[21] https://en.wikipedia.org/wiki/Verity_Stob

Figure 8: Paradigmia

wanted to work using C++ at some point in my career, because C++ is another wonderful thing, but with pedigree.

You know, C++ has pedigree. You say that you can write C++ and automatically you have a higher status in the world of developers.

Now the interesting bit of this part of the story is that this company had coding guidelines, which is fairly common in many places. But the thing is, these people explicitly forbade four features of C++ in their codebase:

- Macros.
- Multiple inheritance.
- Template Metaprogramming.
- Standard Template Library.

I will make a pause here, and I will ask anyone with knowledge of C++, please tell me, what is the point of using C++ without these features? What remains of the language without those?

I understand that you might not want to deal with horrid template metaprogramming error messages, or that your project started when the STL was still young and buggy and Boost did not exist yet, or that you are part of the camp who wets their pants when you mention the words “multiple inheritance.”

Boooooooo. Scared, huh? Everybody is scared of multiple inheritance, yet very few have actually tried it. Multiple inheritance is absolutely awesome. It is like a drug. It is a one way street. And in C++ you can choose public or private inheritance. That gives you lots of possibilities to crash in different ways. I like having options for my funeral.

```
  required from '_IIter std::find(_IIter, _IIter, const _Tp&) [with _IIter = _
  required from here
error: no match for 'operator==' in '__first.__gnu_cxx::__normal_iterator<_Ite
note: candidates are:
note: template<class _T1, class _T2> bool std::operator==(const std::pair<_T1,
note:   template argument deduction/substitution failed:
note:   'std::vector<int>' is not derived from 'const std::pair<_T1, _T2>'
note: template<class _Iterator> bool std::operator==(const std::reverse_iterat
note:   template argument deduction/substitution failed:
note:   'std::vector<int>' is not derived from 'const std::reverse_iterator<_I
note: template<class _IteratorL, class _IteratorR> bool std::operator==(const
note:   template argument deduction/substitution failed:
note:   'std::vector<int>' is not derived from 'const std::reverse_iterator<_I
note: template<class _T1, class _T2> bool std::operator==(const std::allocator
note:   template argument deduction/substitution failed:
note:   'std::vector<int>' is not derived from 'const std::allocator<_T1>'
note: template<class _Tp> bool std::operator==(const std::allocator<_Tp1>&, co
note:   template argument deduction/substitution failed:
note:   'std::vector<int>' is not derived from 'const std::allocator<_Tp1>'
note: template<class _Tp, class _Alloc> bool std::operator==(const std::vector
note:   template argument deduction/substitution failed:
note:   mismatched types 'const std::vector< Tp,  Alloc>' and 'const int'
```

Figure 9: The beauty of C++ template error messages

The worrysome part of this is that without these things, particularly without multiple inheritance, C++ is worst than Java; because if you think about it, Java interfaces (as well as C#, Kotlin and PHP interfaces, and Objective-C and Swift protocols) are all lightweight, more controlled forms of multiple inheritance; technically, a protocol or an interface is nothing else than an abstract class with pure virtual methods. You can totally use C++ classes as interfaces, if you want to, and it can be a good idea, just like interfaces are a good idea in Java, PHP, C#, Swift, Objective-C and Kotlin.

So what do you do without it? Well, you repeat code all over the place.

Because for these people, no, multiple inheritance was bad, so their system consisted of large, very large families of classes using single inheritance, with tons of duplication of code nearly everywhere.

This was half a million—you heard right, five hundred thousand lines of code, which took around 3 hours to compile in its entirety. The automated test suite (around 4000 tests) used to take 5 hours to run and generated tons of Excel files, which were then loaded into VBScript programs to verify that they contained the values expected by each calculation.

One of the coding guidelines actually stated: please do not select the “Clean” option in Visual Studio if not needed, as the compilation process is really long.

We were using Visual Studio 6, the version from 1998, because Visual

Studio .NET 2005 (the version currently available at that time) was strictly unusable with C++ projects back then.

Living in 2006 developing software just like they did back in 1989. Including CVS. This is not a joke. Linus was barely starting to work in Git by that time. Subversion was available but many shops were not confident about using it yet.

4. Objective-C (aka Coolia)

Figure 10: Coolia

Objective-C is a wonderful thing. But I love it, for a very simple reason: it is not obnoxious.

Some people need their programming languages to be really obnoxious. “Oh this not an array of string, I will not compile this.”

I divide programming languages in two families. East Coast and West Coast.

This division has to do with my own perception of America. You know, you have on one side the East Coast, kinda uptight, New England, Connecticut, the founding fathers, the Mayflower and the Tea Party and all that. The programming languages of the East Coast are very uptight and demand that you check everything before they compile anything. Look at the error messages that a typical C++ compiler throws at you when you try to use template metaprogramming incorrectly. Exactly the same thing that scared the shit off the people in my previous job.

C++, Java and Swift are languages of the East Coast. They drink their tea with scons at 5pm and gossip about how PHP, this formerly

homeless, drunk language is getting good at business and might open a lambda shop anytime soon. It even has traits, they say. Oh dear!

Pascal is a very old member of this group, flying in from Zürich every so often. The poor guy is retired now, living its days in a clinic in the Alps, to treat its schizophrenia.

Just like any wealthy New Yorker, C++ was a fashion victim, and every ten years would change its wardrobe and look different every time. These days it is all about concepts and lambdas and type inference and whatnot.

```
external static auto auto(auto &ref) [&auto] {
    return reinterpret_cast<auto> (auto) auto;
}
```

Together, the East Coast languages would compile the world every so often, making sure that everybody could seat in a `Seat<Language<EastCoast>>` and that everybody had a `Cup<Tea>` in their hands.

On the other hand, Objective-C was a West Coast language. Objective-C is a hippie. You throw stuff at it and it just trusts you. "Peace maaaaaannn... if you say this is an `NSArray` I believe you..." and it leans back in its armchair, smoking weed, listening to Bob Marley, Pink Floyd, The Doors, or Janis Joplin, looking at you with deep eyes, telling you stories about Buddhism and how important is to take care of ecology, and that tonight they are going to a retreat in the beach to pray for Yemanjá and the salvation of dolphins in Nicaragua.

The West Coast language group was founded by Smalltalk, an old guy from San Francisco who used to quote Hunter Thompson and Alan Watts in between LSD pills.

Ruby joined the group in the early nineties; a wise young Japanese language who had traveled a lot around Europe by train, and who worshipped Smalltalk and followed its steps very closely.

JavaScript is another emeritus member of this club, borrowing some traits from Objective-C and some others from a third, obscure group of languages, founded by Lisp, Haskell, Clojure, and its relatives.

(As a side note, it is worth mentioning that the whearabouts of this third group of programming languages remain hard to map, because their meetings never have side effects, they filter and reduce their members continuously, and their facial expressions never betray any global state.)

VBScript used to hang out sometimes in those retreats of the West Coast, but everybody knew that it was the kid of a very rich family and nobody wanted to talk to him. Objective-C looked at VBScript with compassion, because it had a secret himself, too.

Objective-C had a cousin in the East Coast. And the cousin was none other than C++ himself.

I know, I know.

Objective-C and their cousin secretly shared a beautiful secret, something that nobody considered important at first, but turned out to be fantastic.

They shared an Application Binary Interface.[22]

In spite of their obvious differences, they both were kind enough to each other, they generated code that was compatible with that generated by their cousin, and they could link each other with grace and compassion.

But nobody, neither in the West nor in the East Coast, knew about this, because it would have been a terrible blow to their respective reputations.

5. JavaScript (aka Undefinedia)

Figure 11: Undefinedia

At some point in my career I thought I was a good idea to build apps using a weird combination of Sencha Touch, PhoneGap (or Cordova or whatever the name) and all written with this language called JavaScript.

Heck, I even wrote two books about the subject. Two. With 5-star reviews on Amazon and everything.

JavaScript is another wonderful language from the West Coast, even more than VBScript because at least it has a book named “The Good Parts” that has a positive page count.

[22] https://en.wikipedia.org/wiki/Application_binary_interface

JavaScript followed the recommendations of Ruby and also traveled to Denmark; there he met Lars Bak and they had together V8, which later brought us Node.js and Cordova and Electron.js and plenty of other wonderful things.

And then Electron.js and Cordova apps became the Visual Basic and .NET apps of today, requiring them to bundle a 30 MB runtime at each build.

JavaScript became so popular after its trip to Europe that many other languages started compiling into it, hoping to get a mermbership card from the East Coast group of languages; CoffeeScript, LiveScript, TypeScript, Kotlin, UberScript, ToffeeScript, Mascara, Objective-J, Script# and many, many more.[23]

They all wanted to be the ones who would unify both language groups, with a big feast, where all languages, including Smalltalk and C, would be reunited at last, sharing a common interface, all binary and portable and fast and true. But you know how bitchy languages can be, and this never happened.

6. Swift (aka Migrania)

Figure 12: Migrania

[23] https://github.com/jashkenas/coffeescript/wiki/List-of-languages-that-compile-to-JS

One of the most brilliant aspects of both the JVM and the CLR is that they are, by definition, Application Binary Interfaces[24]. They are ABIs. They define a common binary format that other languages can compile into, and the platforms guarantee that this format will be understood and executed.

Even COM, the component system on top of which VBScript was built upon, was "a binary-interface standard for software components introduced by Microsoft in 1993. It is used to enable inter-process communication and dynamic object creation in a large range of programming languages."—dixit Wikipedia.

Funny story, I once left a comment in an article about COM in CodeProject.com[25] saying that you could actually use COM in Mac OS X[26]… there was even an article in the O'Reilly Mac Dev Center. How about that.

All this means that in COM, Java or .NET, you can take your language, any language, and with a few modifications (most of them regarding memory management and layout of objects) you can create binaries for them, and benefit from large amounts of pre-built code, known as COM components, or the Java Class Library, or the .NET Framework Class Library.

This is what brought us F#[27], JRuby[28], Kotlin[29], Scala[30] (and, yes, also VBScript) and so many other cool things. Both the JVM and the CLR are high-level, portable virtual machines that even clean after your language runs, so that no objects are left unattended in the heap of your application. Is not that fantastic?

Even better, the specifications of both are (to a certain extent) open and standardized, and they are available for a variety of hardware architectures and operating systems. You can write a "Hello World" in Java and (again, to a certain extent) you can run the compiled thing in any other platform. JARs containing Struts or Java Server Pages applications built in 2002 can still run in JBoss or Tomcat today, either in Windows, macOS or Linux. The CLR is slowly getting into other regions as well, and with Xamarin[31] you can even transpile those .NET assemblies you built with C# into native executables for iOS.

The name of the LLVM project[32] means "Low Level Virtual Machine." And Swift is arguably the first language born from the project, one

[24] https://en.wikipedia.org/wiki/Application_binary_interface

[25] https://www.codeproject.com/Articles/633/Introduction-to-COM-What-It-Is-and-How-to-Use-It?msg=823146#xx823146xx

[26] http://www.macdevcenter.com/pub/a/mac/2004/04/16/com_osx.html

[27] http://fsharp.org/

[28] http://jruby.org/

[29] https://kotlinlang.org/

[30] http://www.scala-lang.org/

[31] https://dotnet.microsoft.com/apps/xamarin

[32] http://llvm.org/

whose basic premise was the ability to interoperate with whatever we had before; that is, Objective-C and Cocoa.

The Swift project started in 2010. The language was released in 2014. We are in year 2017 now, and apparently Swift will not feature an ABI anytime soon, at least not this year. But there is a manifesto for its stability[33]. At least that.

In this case, just as in many other situations, **the best is the enemy of the good.** I, for one, am tired of waiting for the language to stabilize and play well with Objective-C and maybe other languages.

I am tired of migration assistants in Xcode. I am tired of being told "file radars" as a polite way to tell me to f*** off. I am tired of Xcode exploding in my face every five minutes. I am tired of iCloud being unreliable. I am tired of iTunes being such a CPU and memory hog. I am tired of merging Storyboards with Git. I am tired of the Touch Bar. I am tired of the dongles. I am tired of the Apple TV Remote. I am tired of all this courage.

> Apple TV remote. Mac App Store. Mac Pro 2013. Code signing. Apple Maps. File a radar. Touch Bar. Swift ABI. Courage. iTunes. Xcode. Mac.
>
> — akosma (@akosma) April 16, 2017[34]

What's the French word for "tired"? Ah, je suis fatigué.

And one year we have code documentation using reStructuredText from the Python world and the next year is Markdown. Yes, that same Markdown by Gruber we have seen before.

7. Local Cluster (aka Here, Now)

I have changed galaxies quite often in my career.

As I said last year[35]:

> So my recommendation to you is: choose your galaxy wisely, enjoy it as much or as little as you want, but keep your telescope pointed towards the other galaxies, and prepare to make a hyperjump to other places if needed.

Many of the programming languages I have used in the past have new versions out, and even better, that they were all converging and somehow starting to look the same: C++17, PHP 7, C# 6, ES 6, Java 8, Kotlin 1.1, Swift 3.1…

> C# 6 PHP 7 ES 6 C++17
>
> time of renewal for many classic prog langs these days, huh?

[33] https://github.com/apple/swift/blob/master/docs/ABIStabilityManifesto.md
[34] https://twitter.com/akosma/status/853617911252045825?ref_src=twsrc%5Etfw
[35] /blog/being-a-developer-after-40/

Figure 13: Local Cluster

> what I find interesting is their convergence.
>
> — akosma (@akosma) April 3, 2017[36]

Boom.

> Luckily C++ is just Swift with required semicolons, a simpler type system, and a commitment to source compatibility. You're going to love it
>
> — Callionica (@Callionica) April 9, 2017[37]

You're going to love it.

Mainstream programming languages are very similar to one another these days. The West and the East Coast groups are approaching from one another.

Pretty much all languages have lambdas[38] these days.

Java and C++ have gotten optionals recently, also called "option types," "Maybe" or "Nullable" just as many other languages[39].

C++ got type inference exists 2011, and it got better in Java 8.

PHP & Scala have traits; Java 8 has default methods; Ruby has mixins; Swift has protocol extensions; C++ has had multiple inheritance since day one; and Kotlin has optional interface implementations.

Both Scala and Kotlin have object declarations.

[36] https://twitter.com/akosma/status/848970460876472320?ref_src=twsrc%5Etfw
[37] https://twitter.com/Callionica/status/851068870626426880?ref_src=twsrc%5Etfw
[38] https://en.wikipedia.org/wiki/Anonymous_function#List_of_languages
[39] https://en.wikipedia.org/wiki/Option_type#Names_and_definitions

Objective-C has categories, Swift has extensions, Kotlin has type extensions, C# has method extensions for partial types, and in JavaScript, well, you can add stuff in `prototype` and all that.

In galactic terms, galaxies are much closer than before, and our hyperspeed jumps are cheaper and easier than ever.

But we might as well be reaching a singularity at the same time.

> It's a crazy world when THIS is my Windows 10 Desktop developing @dotnet[40] pic.twitter.com/Js8wgmx7ER[41]
>
> — Scott Hanselman (@shanselman) April 13, 2017[42]

Crazy! Geez.

8. Guardians Of The Galaxy (aka You)

Figure 14: I know, I might be breaking some copyright here. Enjoy while it lasts.

One thing that is common to all galaxies is, sadly, the situation of many of our fellow workers.

Micromanagement.

Lack of accessibility.

Discrimination.

[40] https://twitter.com/dotnet?ref_src=twsrc%5Etfw

[41] https://t.co/Js8wgmx7ER

[42] https://twitter.com/shanselman/status/852398546623971328?ref_src=twsrc%5Etfw

Pseudoscrum.

Open Spaces.

Harassment.

Lack of testing and quality.

Burnout and depression.

Figure 15: Πάντα χωρεῖ καὶ οὐδὲν μένει (Ἡράκλειτος) or “Plus ça change, plus c’est la même chose” (Karr) ?

In every galaxy I visited, women and groups of humans other than white males between 25 and 35 years old are harassed, dismissed and discredited. Which means that by large, our industry sucks more than any shitty programming language.

We, developers, are the ones getting the workload and are the ones actually burning out, and it does not need to be like that.

We have to start taking back the ownership of our craft, of our code, of our pride. This means breaking the typical employer—employee relationshp, and start choosing jobs where we can take ownership (as in literal and as in Scrum) of the code we produce.

We must stop releasing code that is uncovered by tests.

We must stop accepting open spaces as the default setup for our offices.

We must stop developing unaccessible apps.

We must stop accepting to work in jobs without inclusion and diversity.

We must stop accepting jobs in companies that destroy privacy and violate human rights.

Just flatly refuse to work in those environments. Because those environments are the ones that are making our code such a mess. It is not the choice of the programming language, the name of your favorite design patterns or the choice of spaces vs. tabs; those are just details, smoke screens blinding us from the truth.

Remember that in our industry there is still more demand than offer in the developer job market; you have the power to leave offending workplaces and even better, you can start your own adventure—which I strongly recommend you to do.

The galaxy we have to fight for is the one that is located everywhere, the one where we all belong, even without knowing it. The one that calls us like a big home, where we can actually work differently, and

build a better world, where ethics are not an afterthought, where accessibility is a default.

I call to you, to every one of you, to change your own environment, your own surroundings, your own team, your own persona, so that we can actually, finally jumpstart this Age of Aquarius[43] everybody was talking about at the turn of the century.

We were supposed to become a better species.

We were supposed to become better developers.

We were supposed to make this world a better place.

Figure 16: Change the world, they said.

It is not the programming language.

It is not the frameworks.

It is not the patterns.

It is us.

The most important thing is the trip itself, not the destination.

Thanks a lot for your attention.

[43] https://en.wikipedia.org/wiki/Age_of_Aquarius

Rogelio Suárez y La Vida Peligrosa

Adrian Kosmaczewski

2017-08-14

Acabo de publicar mi primer novela en castellano, titulada “Rogelio Suárez y La Vida Peligrosa”. Esta disponible en Leanpub[1]. El arte de tapa y las ilustraciones dentro del libro son obra de la increíble y talentosa Yohanna Etchemendy[2].

Escribí el primer borrador en Daedalus Touch para iPad, entre el 10 y el 26 de febrero del 2016. Se pueden bajar un capitulo gratuito aqui mismo[3].

Una entrevista televisada a un ignoto personaje argentino, un tal Rogelio Suárez, que vivió en Chubut, Santiago del Estero y ahora en el Gran Buenos Aires. Una serie de pensamientos, poesía en forma de prosa y prosa en forma de poesía. Una secuencia de viñetas sobre el ser argentino. Una reflexión sobre una vida pasada. Una proyección a lo que fue, a lo que podría haber sido, a lo que nunca se sabe si será o no.

La vida de Rogelio no es extraordinaria, no es digna de tal despliegue televisivo, es de una anonimidad atroz, sin embargo es en esa banalidad que está toda su universalidad. Rogelio está en cada uno de nosotros. Entrevistarlo no es otra cosa sino conocerse a sí mismo.

Prepárense para un viaje hacia ustedes mismos. Gracias por estar ahí, Rogelio.

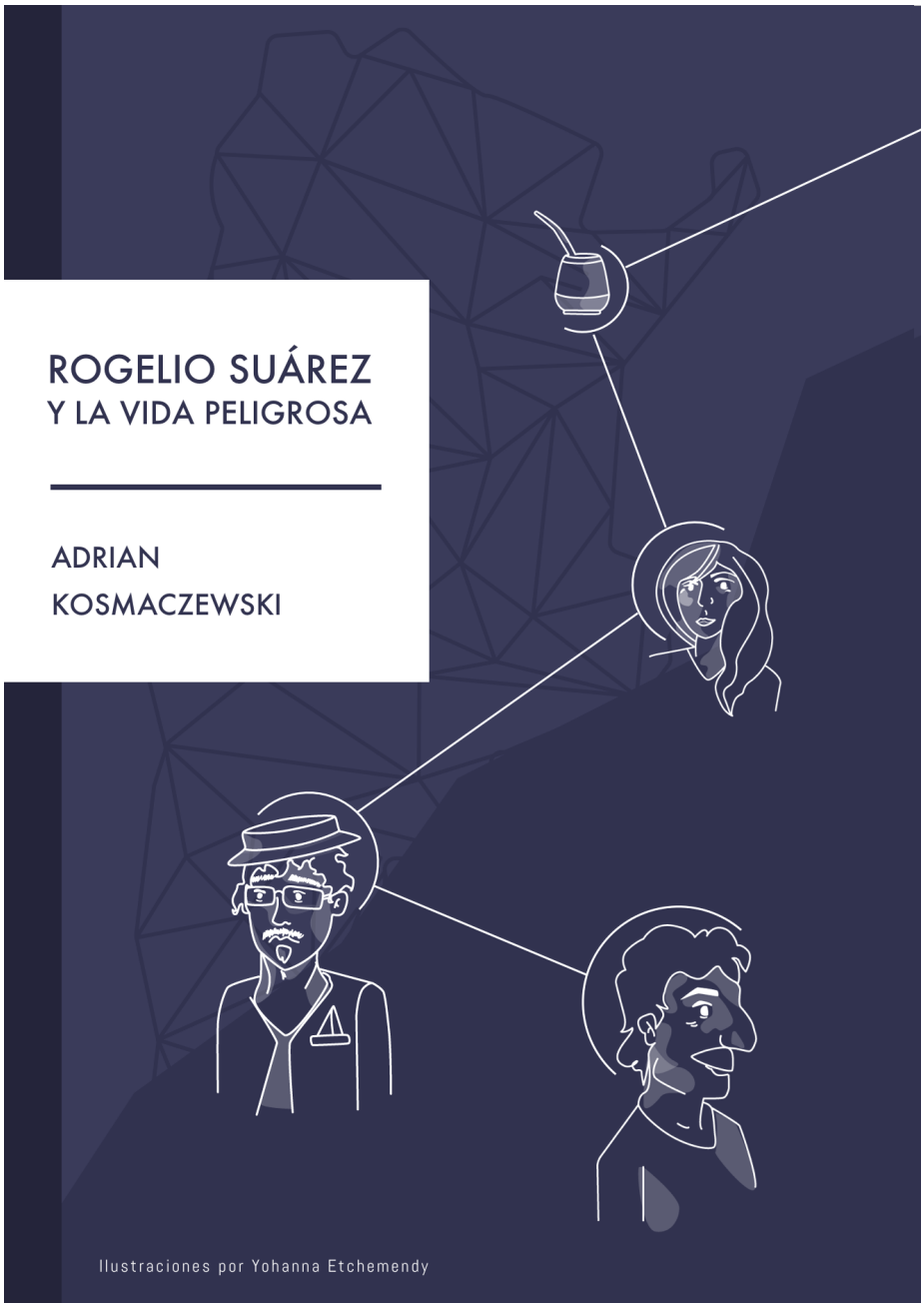
No, gracias a vos.

---

[1] https://leanpub.com/rogelio-suarez-y-la-vida-peligrosa/

[2] https://www.linkedin.com/in/yohannaje/

[3] rogelio-suarez-y-la-vida-peligrosa-sample.pdf


ROGELIO SUÁREZ
Y LA VIDA PELIGROSA
ADRIAN
KOSMACZEWSKI
Ilustraciones por Yohanna Etchemendy

Docker - Didapro 7 - DidaSTIC

Adrian Kosmaczewski

2018-02-06

Bonjour, et bienvenus a cet atelier a propos de Docker chez Didapro.

Je m'appelle Adrian Kosmaczewski, et je travaille comme développeur d'applications mobiles et aussi comme enseignant de programmation dans le privé. J'ai aussi publié 5 livres sur le sujet de la programmation mobile et je parle regulierement dans des conferences partout dans le monde.

J'ai prevu une session de questions et reponses a la fin de cette session, donc je vous prie de garder vos questions pour la fin. Cette presentation et tout le materiel que je vous montre sera disponible gratuitement sur Github, pour que vous puissiez suivre cette presentation sans devoir prendre des notes.

Le probleme

Bien plus important, depuis 2003 je donne regulierement des cours et des ateliers a propos de la programmation; j'ai enseigne Java, C#, JavaScript, Objective-C, Swift, Kotlin. Depuis toujours il y a un probleme qui revient chaque fois que je commence un nouveau cours: faire en sorte que tous les etudiantes et etudiants aient la meme configuration de systeme d'exploitation, langages de programmation et librairies pour suivre ce que je fais sur l'ecran de mon portable.

Generalement mes cours sont au format "workshop" ou les 10 ou 12 etudiant(e)s que je recoit suivent en direct les manipulations que je fais sur l'ecran, et ensuite je m'assure que le code "compile et tourne" sur chacune des machines. Ensuite je donne les elements de theorie qui correspondent, et je repete cette boucle quelques fois pendant deux ou trois jours.

Il est donc tres important pour moi que tout le monde puisse suivre, et cela veut dire imperativement que tout le monde ait les memes outils installes de la meme facon sur leurs portables. Pendant des années j'ai cherche a resoudre ce probleme de plusieurs facons, en utilisant des outils comme:

- Des instructions precises via e-mail quelques jours auparavant de la session; le probleme de cette approche c'est que les gens ne lisent pas toujours ce qui y est ecrit. Donc, il faut mettre de

cote un peu de temps, le “lundi matin” du cours, pour etre sur que tout le monde a le setup de base pour travailler, et que tout le monde pourra y participer.
- Telecharger une machine virtuelle, par exemple pour VirtualBox; le probleme avec cette approche c’est la question des licenses, car on ne peut pas donner comme ca une copie de macOS ou Windows; aussi il y a le probleme de la taille des fichiers, qui peut être conséquente, et aussi le fait de mettre à jour ces machines virtuelles, qui est un peu difficile.

Aussi, il faut dire que le monde du developpement logiciel d’aujourd’hui est vraiment complexe; par exemple, pour enseigner le developpement web avec un stack comme “MEAN” (Mongo, Express.js, Angular et Node) sur des portables Mac (ce qui n’est pas deraisonnable ces jours-ci), il faut installer, dans l’ordre:

1. Homebrew
2. Node & npm
3. Mongo
4. Express
5. Angular
6. Des lib de testing, comme Jasmine, Karma ou Mocha
7. Un editeur de texte, par exemple WebStorm ou Visual Studio Code
8. Un terminal comme iTerm2 pour y etre plus confortable, avec d’autres outils comme htop, tmux, etc

Et on peut y ajouter des couches! Par example si ensuite vous voulez montrer comment faire des applications mobiles sur le stack MEAN vous devriez y ajouter su ionic, et on pourrait encore continuer.

Le probleme, je suis sur, ne vous est pas etrange, et dans mon cas je n’ai que 12 etudiant(e)s! Je ne voudrais pas m’imaginer ce que ca donne avec une classe de 30 personnes.

Idealement, il me fallait une plateforme avec les caracteristiques suivantes:

- Legere a installer et distribuer a mes eleves
- Rapide a démarrer et eteindre
- Libre de droits
- Cross-platform Windows, Linux et Mac
- Facile a mettre a jour pour les prochaines sessions de cours
- Eventuellement, facile a passer a mes collegues enseignant(e)s aussi si besoin etait.

Une solution: Docker

Depuis peu j’utilise Docker pour resoudre ce probleme; Docker est une technologie relativement recente, tres a la mode (et avec raison!) dans le monde des startups.

Pour resumer ce qu'est Docker, c'est un systeme de machines virtuelles legeres, qui se base sur certaines proprietes et librairies du noyau Linux. Les machines virtuelles Docker ne sont pas completement isolees de leur hote, et peuvent partager des resources comme le kernel ou des librairies. Pour cette raison, elles démarrent presque instantanément et n'ont besoin que d'une fraction des resources requises par une machine virtuelle "traditionnelle" comme VirtualBox ou VMWare.

En plus, Docker et 100% gratuit et libre, et tourne aussi sur Mac et Windows. La compagnie derriere Docker fait de l'argent avec du support pour entreprises et des produits "pro" dont je ne connais pas les caracteristiques, ne les ayant jamais utilises dans mon travail.

Images et Containers

Pour comprendre Docker, il faut connaitre la terminologie qui y est associée: **les Images, la Registry et les Containers.**

Les machines virtuelles Docker sont crées et distribuées dans la forme de Images. Une image est la representation en disque d'une machine virtuelle Docker, de la meme facon qu'une image de disque "ISO" represente les contenus d'un CD de musique ou un DVD.

Pour creer une image, il faut utiliser un fichier au format particulier appele un `Dockerfile`. On va apprendre plus sur ce fichier dans quelques instants. Il faut ensuite faire tourner une commande Docker sur un terminal, `docker build` en y passant le chemin d'acces d'un Dockerfile.

Lorsque l'on cree une image, elle est ajoutee au Registry local. C'est le deuxieme terme important de Docker. Une Registry c'est un repository d'images. On peut utiliser une registry pour partager des images avec d'autres utilisateurs. Docker offre gratuitement une image Docker (bien sur!) qui peut etre utilisee pour faire une Registry locale dans votre institution, ecole ou college. Avec une registry, vous pouvez reutiliser des images et en creer des nouvelles qui se basent sur celles deja existantes.

Lorsque l'on démarre une image Docker, on obtient un Container, qui est une entite en cours d'execution de cette image. C'est ici que vous aurez vos propres services, applications, et librairies pretes a l'emploi. On peut lancer plusieurs containers separes a partir de la meme image; en fait, la seule limite pour cela est la quantite de memoire vive disponible.

Differences entre VM et Docker

Pour comprendre la difference entre une VM "classique" (VMWare, Parallels ou VirtualBox) et Docker, les graphiques suivants peuvent aider.

Les principales differences entre les deux peuvent etre resumees ainsi:

- Chaque instance d'une VM doit charger tout un systeme d'exploitation complet, ce qui peut etre relativement lourd avec des systemes avec interface graphique. Les containers Docker, au contraire, ne chargent qu'une partie du systeme Linux, utilisant le kernel du hôte pour la plupart des taches.
- L'isolation d'une VM et complete; les applications y tournent sans avoir, a priori, aucune connexion avec le hôte, hormis des connexions reseau et des drivers video ou peripherique. Les containers Docker, eux, ne sont pas completement isoles, et peuvent avoir un acces plus large a la machine hote.
- Le temps de demarrage d'une VM est generalement long; il faut compter quelques minutes parfois. Un container Docker ne prends que quelques millisecondes a démarrer, et les applications peuvent y tourner plus vite.
- Les containers Docker ne peuvent pas "emuler" d'autres architectures logicielles; c'est a dire que l'on ne peut pas avoir des containers Docker qui tournent du code x64 sur un hôte avec une CPU de type ARM.

Creation d'images

Comme mentionné precedemment, pour créer une image Docker, on va devoir creer un fichier appele `Dockerfile`. Ces fichiers comportent des instructions tres simples a lire et a comprendre, qui vont faire des changements precis, couche par couche, a une machine de base, qui peut ne pas etre disponible sur le hote en question.

La premiere commande d'un `Dockerfile` est la commande `FROM` qui specifie l'image de base qui va etre utilisee pour construire une nouvelle image. Pour resoudre le probleme de la poule et les oeufs, Docker propose une image appellee `scratch` qui est un noyau Linux minuscule avec le minimum necessaire pour faire tourner une image.

Une image de base tres habituelle est `ubuntu:16.04`, dont le nom décrit bien les contenus. Tres populaire aussi, la distribution Arch Linux est souvent utilisee comme image de base, grace a sa capacite de mettre a jour automatiquement les packages ajoutes, ce qui en fait un choix interessant pour services connectes en permanence.

Apres on y ajoute une commande `LABEL` qui a pour but d'ajouter des metadonnees qui seront utilisees para la Registry pour indexer et chercher des images, et qui permettent a d'autres utilisateurs de comprendre ce que fait votre image.

Ensuite, on utilise des commandes comme `RUN` pour faire tourner des programmes dans l'image; typiquement la premiere commande est toujours celle requise pour mettre a jour le gestionnaire de packages de votre distribution Linux:

RUN apt-get update RUN apt-get install -y build-essential

Ensuite il y a d'autres commandes, comme `COPY` pour pouvoir y ajouter des fichiers arbitraires dans l'image, ou `WORKDIR` pour changer de directoire a l'interieur de l'image.

Tout cela permet de creer des images assez complexes, avec meme des utilisateurs et une grande variete de variables d'environnement pretes a l'emploi.

Demo

On va voir maintenant un petit movie (fait avec Asciinema) qui montre les etapes et les mutations requises pour faire une petite image Docker.

Imaginons que vous devez enseigner la programmation en C. Oui, je sais, c'est pas forcement votre tasse de the mais c'est un exemple tres simple pour vous montrer comment faire.

L'idee ce sera de creer une image Docker avec les outils de base de developpement en C, pour compiler une petite application en C, le desormais classique "Hello World".

On va donc se baser sur Ubuntu 16.04, et on va y mettre a jour le systeme de gestion de packages. Ensuite on y installera les outils de developpement C de base, et finalement on va y creer un folder appele "build". Dans ce folder on y copiera notre code source, et un Makefile pour nous faire la vie plus simple.

Ensuite, on tape la commande pour faire construire notre image:

```
docker build --tag akosma/helloworld .
```

Le point a la fin de la command fait reference au folder courant, qui doit contenir imperativement un `Dockerfile`. Le `tag` qui est passe dans la commande ci-dessus donne un nom a l'image, pour qu'elle soit plus facile a reperer entre tant d'autres.

Une fois qu'on a construit notre image, on la retrouve facilement grace a la commande

```
docker images
```

qui montre la liste des images disponibles dans la registry locale. On y voit notamment deux images, celle que l'on vient de creer et aussi l'image `ubuntu` qui reste copie en locale, de telle facon a ce que les prochaines constructions en base a celle-ci se fassent plus rapidement.

On peut creer un container a partir d'une image de facon tres simple:

```
docker run --tty --interactive akosma/helloworld bash
```

Les parametres `--tty --interactive` apparaissent tres souvent dans la litterature comme simplement `-it`, et donnent l'instruction a

Docker de ne pas seulement faire tourner le container en memoire, mais aussi d'attendre nos commandes et de montrer les resultats de facon interactive. La commande qu'on passe a la fin, `bash` indique le terminal de ligne de commande que l'on veut utiliser pour notre session interactive.

On peut aussi passer un parametre `--name` pour donner un nom sympa a notre container.

On peut se promener maintenant a l'interieur de notre container, qui a exactement la configuration que l'on veut: un Ubuntu 16.04, avec les outils de developpement du langage C, et un folder `/build` avec un `Makefile` et notre fichier `hello.c`.

On peut faire un `ls ..` et voir le contenu du systeme de fichiers; il y a tout ce que l'on attends d'une distribution Linux, en ce cas precis, Ubuntu.

Ensuite on tape `make` et on voit, avec `ls` que l'on a cree un executable a partir de notre fichier `hello.c`; on peut faire tourner ce fichier comme n'importe quel autre programme.

Je peux maintenant me "detacher" de ce container, en tapant la combinaison de touches `CTRL + P` et `CTRL + Q` a la suite. Je peux voir maintenant les containers qui tournent:

```
docker container list
```

Cette commande est la meme que `docker ps`. On y voit un "nom" a la fin de l'output, qui est un nom aleatoire que Docker donne a chaque container. Il y a aussi un ID alphanumerique plus complexe, lui aussi aleatoire, qui est le container ID officiel.

On peut "rentrer" dans notre container en tapant la commande

```
docker exec -it angry_kilby bash
```

Et au lieu de `angry_kilby` je peux utiliser le container ID, of course. Je vois que mon fichier executable `hello` est toujours la.

Maintenant je tape `exit` et cela a pour effet de quitter `bash`, mais puisque `bash` et le programme principal de ce container et c'est un container interactif, alors le container aussi il disparait. Un `docker ps` nous montre qu'il est parti maintenant. Par contre, notre image est toujours la.

Si on veut arreter un container sans y rentrer, on peut faire

```
docker stop angry_bird
```

et cela aura l'effet d'arreter le container. On peut y donner un timeout a ce processus (par defaut c'est 10 secondes), apres quoi le container est `killed`.

Bien sur vous pouvez `docker kill` si besoin est, mais ce n'est pas vraiment recommande.

Tres souvent on a besoin de partager des document ou des fichiers entre le container et le hôte. Pour cela, rien de plus simple que d'utiler la commande:

```
docker run -it --volume $PWD:/build akosma/helloworld bash
```

Cette fois-ci on utilise un parametre `--volume` qui prend un parametre en deux parties: la premiere partie `$PWD` fait reference a folder courant dans le hôte, et la deuxieme fait reference a un folder dans le container. Maintenant si on execute `make` dans notre container, et qu'on sort du container, on a un fichier executable Linux `hello` dans le folder courant, mais bien entendu il n'est pas executable car je suis sur un Mac. Le container vraiment est un Linux qui tourne n'importe quel logiciel Linux!

Maintenant on va creer un nouveau container base sur notre image "helloworld". Pour cela on a besoin d'un nouveau `Dockerfile` qui va nous permettre de faire un peu de developpement web en PHP. Pour cela, on va ajouter quelques commandes et reconstruire notre image, en ajoutant la derniere mouture de PHP. A la fin de notre `Dockerfile` on ajoute une command `EXPOSE` pour documenter le fait que l'on exposera le port 4000 au monde exterieur.

Par défaut, les containers peuvent se parler entre eux, mais n'exposent pas leur ports au monde exterieur.

```
docker run -it --detach --publish 127.0.0.1:8000:4000 akosma/php php -S 0.0.0.0:4000
```

Le parametre `--detach` (`-d`) specifie que l'on fait un container "detached" qui commence a tourner sans interactivite. Aussi on specifie le parametre `--publish` (`-p`) pour mapper le port 8000 du hôte au port 4000 du container. Finalement, on lance le serveur web incorporé dans PHP a l'aide de la commande `php -S 0.0.0.0:4000` qui le lance de telle façon qu'il ecoute des requetes depuis n'importe quelle interface. Ajoutez-y une commande `--volume` et la base de données MySQL dans le mix, et vous avez un environnement d'etudes LAMP pret a l'usage!

Outils

Bien sûr, utiliser la ligne de commande n'est pas du gout de chacun. En pensant à cela, Docker a fourni un outil appele Kitematic, compatible avec Windows, Mac et Linux, qui peut etre utilise pour gerer facilement les images et containers Docker.

Aussi, plusieurs editeurs comme Visual Studio Code vous permettent de voir et d'interagir avec vos images et containers directement depuis leur interface, pour pouvoir démarrer ou stopper des containers et en démarrer de nouveaux a partir d'images.

Limitations

Comme tout outil, Docker n'est pas parfait, et a quelques limitations dont il faut tenir compte:

- Un hôte ne peut tourner que des containers de la même architecture; pas de containers PPC ou ARM sur un hôte x64! Ceci est la plus grande difference entre les containers et les machines virtuelles "classiques".
- En revanche, et ceci pour plusieurs raisons, les containers Linux tournent sur tous les hôtes; tant Docker pour Mac que pour Windows ont des "couches logicielles de compatibilite" qui permettent de les faire tourner sans probleme.
- Aussi, il faut savoir que les containers Microsoft ne tournent que sur Windows (of course!). Il est ainsi possible de faire tourner des applications ASP.NET sur des containers Docker qui heritent de l'image `windowsservercore`[1].

Scenarios

Cette demo vous a montre comment faire quelques images, et comment faire quelques containers a partir de ces images. Maintenant on verra d'autres usages possibles de Docker, qui pourraient vous donner des idees.

- Faire tourner Moodle facilement a l'interieur d'un container, sans devoir passer par un serveur ou un departement de IT.
- Enseigner des langages de programmation esoteriques ou historiques, comme LISP, Prolog, COBOL, Pascal ou Fortran, de facon simple et interactive.
- Faire tourner des vieilles versions de logiciels pour les demontrer sans devoir modifier la configuration de la machine hôte.
- Enseigner le developpement web de facon facile, que se soit LAMP ou MEAN, avec les bases de données et les langages de programmation deja installes et prets a l'usage.
- Enseigner Docker! C'est une technologie tres en vogue, tres prisee par l'industrie, qui est devenue un standard en tres peu de temps. Habituer vos eleves a utiliser Docker, ne serait-ce qu'a en connaitre les principes de base, c'est deja un grand pas en avant pour leur insertion laborale.

Recommendations

Ayant vu quelques uns des avantages de Docker, on peut passer maintenant a quelques recommandations pour vos images en milieu educatif:

- Utilisez toujours Linux pour vos images, dans la mesure du possible, car elles sont les plus portables de toutes.

[1] https://hub.docker.com/r/microsoft/windowsservercore/

- Creez un “arbre” d’images, avec une image de base que vous pouvez mettre a jour regulierement.
- Automatisez la creation et la mise a jour des images avec un script, car il est facile d’oublier les commandes Docker apres un temps.
- Si vos images sont interactives, je vous recommande d’utiliser une image plus complete que celle de Ubuntu 16.04, comme par exemple la Phusion Base Image[2] qui est plus complete et mieux configuree pour l’usage interactif.
- Si vous avez besoin de partager des applications interactives, ajoutez un serveur de VNC dans votre image et connectez-vous a celle-ci avec un viewer VNC, disponible nativement sur le Mac.
- Il y a une enorme quantite d’images officielles gratuites disponibles sur Github[3] et aussi quelques unes gratuites mais commerciales (aussi payantes) sur le Docker Store[4].
- Livres: je vous recommande The Docker Book[5] (independant) et le Docker Cookbook[6] de chez O’Reilly.
- Docker propose lui meme de la documentation[7] de grande qualite, et des videos gratuites[8] pour apprendre plus sur cette technologie.

Conclusion

Docker est une technologie nouvelle, ouverte et gratuite, qui est deja devenue un standard de l’industrie. Il est possible de faire tourner des containers sur Windows Azure (le cloud de Microsoft) ou sur Amazon, simplement et avec un clic d’une souris. Plusieurs entreprises suisses deja ont adopte ou sont a point d’adopter Docker pour simplifier leur infrastructure, donc il y a un interet pedagogique et commercial pour exposer les eleves a cette nouvelle technologie, pour qu’ils soient au courant et en fasse la preuve.

Elle peut etre vraiment utile pour standardiser le setup des composants necessaires pour un cours, tout en facilitant la mise a jour et le partage de ces images avec de collegues ou avec la societe toute entiere.

[2] http://phusion.github.io/baseimage-docker/
[3] https://github.com/docker-library/official-images
[4] https://store.docker.com/
[5] https://dockerbook.com/
[6] http://shop.oreilly.com/product/0636920036791.do
[7] https://docs.docker.com/
[8] https://training.docker.com/

A Quest For A Better World

Adrian Kosmaczewski

2018-04-18

This is the presentation I gave at the third App Builders Conference[1] in Lugano, Switzerland, April 17th, 2018.

After having explored personal growth in “Being a Developer at 40”[2], and professional change in “The Developer Guide To Migrate Across Galaxies”[3], I will delve into the third installment of my manifesto for software developers, hoping for us to tackle the larger problems of our modern society with passion, hope and, yes, some dark humor, too.

Introduction

Back in 2016 I took the stage in Zurich and I looked at the past. I looked at my career, at Clippy, at the programming languages that I used, at the managers I had to endure, but most importantly I thought about all the colleagues I had through the years, some of which were in that room that day.

Last year I spoke in Lausanne, and I reflected on the present. The technology change, the craziness around us, the broken debugging tools, and all the little things that make our developer life a chore and a joy.

This year, however, I will project myself to the future. And because past, present and future are all intertwined, in 2016 I could not avoid talking about 2036, and in 2017 I could not stop talking about the past.

We all know that talking about the future of our profession is most often an exercise in abstraction and failure; however, in the current context of protest and mutation, it is simply a pragmatic exercise of enumeration and connection.

> I’ve seen things you people wouldn’t believe. Gopher, Netscape with frames, the first Browser Wars. Searching for pages with AltaVista, pop-up windows self-replicating,

[1] https://appbuilders.ch

[2] /blog/being-a-developer-after-40/

[3] /blog/the-developer-guide-to-migrate-across-galaxies/

> trying to uninstall RealPlayer. All those moments will be lost in time, like tears in rain. Time to die.
>
> — irwin (@irwin) November 22, 2017[4]

1. On The State Of Things

I woke up last Saturday learning that missiles are once again killing innocents somewhere, and that once again they will be just considered a “collateral damage.”

> pic.twitter.com/KaaYzrNeJl[5]
>
> — Ari Paul (@AriDavidPaul) April 14, 2018[6]

In the meantime, the CEO of the biggest social network, one so big that it even had a blockbuster movie made for it, is appearing wide-eyed in front of congressmen in Washington, because his force of good has been used to make somebody the president of some country and that, if one believed the polls before the election, it should not have happened.

Politicians have clearly lost all touch with reality and try to compensate their technical incompetence and ignorance with absolutely absurd decisions, without any sense, as Ancilla van de Leest[7] says, and rightly so.

> European Parliament proposes to grant personhood status to robots. □♀ Experts say nah https://t.co/JD0Ea2hGAr
>
> — Ancilla (@ncilla) April 15, 2018[8]

I also read that people are dying all over the planet because of totally and absolutely reversible causes, but they do not make the news because the war is going on.

> The tech industry is causing so much suffering to the people in it, using it, and indirectly affected by it. This is all connected, and only by valuing human beings and putting compassion first can we hope for any kind of peace.
>
> — April Wensel (@aprilwensel) April 9, 2018[9]

That black americans cannot drive safely in their own country without being pulled over without reason.

---

[4] https://twitter.com/irwin/status/933176066272718850?ref_src=twsrc%5Etfw

[5] https://t.co/KaaYzrNeJl

[6] https://twitter.com/AriDavidPaul/status/984999126571118593?ref_src=twsrc%5Etfw

[7] https://twitter.com/ncilla/

[8] https://twitter.com/ncilla/status/985470502125916160?ref_src=twsrc%5Etfw

[9] https://twitter.com/aprilwensel/status/983214355536756736?ref_src=twsrc%5Etfw

That companies now have a position called the “Chief Conscience Officer” (CCO.) It’s a thing.

That Mike Monteiro[10] is asking for Jack Dorsey to be fired as CEO of Twitter, and rightly so.

That Aral Balkan[11] keeps warning us about Surveillance Capitalism, and rightly so.

> It’s not AI vs people It’s not Big Data vs people It’s not robots vs people
>
> It’s corporations vs people.
>
> — Aral Balkan (@aral) February 13, 2018[12]

That the people in Flint, Michigan, still do not have drinking water without lead in their pipes.

That presidents all over the planet become defacto lifetime dictators, being voted once and again and again and again, never leaving power ever again, and becoming drunk kings drowning in their own corruption, crying for “witch hunts” and silencing opposition.

That women are paid less, harassed, raped, bullied, molested, killed, and forbidden by males to decide over their own uterus, over their own body.

That corporations are using all they know about us to manipulate us, our thoughts, our future.

> “We cannot have a society in which, if two people wish to communicate, the only way that can happen is if it’s financed by a third person who wishes to manipulate them.” – Jaron Lanierhttps://t.co/HYi9ga5i9f
>
> — Florian Albrecht (@flo_muc) April 14, 2018[13]

Meanwhile, in our industry we keep teaching how to reverse linked lists to students, instead of instead teaching them to solve the outstanding inclusion issue that our profession suffers from.

> #comic[14] pic.twitter.com/g0QPauqlZz[15]
>
> — Skeleton Claw (@skeleton_claw) April 2, 2018[16]

That we are building a future of Uber drivers driving each other, of podcast hosts hosting each other, of selfies with celebrities that nobody knows.

---

[10] http://mikemonteiro.com/

[11] https://ar.al/

[12] https://twitter.com/aral/status/963552026679529472?ref_src=twsrc%5Etfw

[13] https://twitter.com/flo_muc/status/985263238312013825?ref_src=twsrc%5Etfw

[14] https://twitter.com/hashtag/comic?src=hash&ref_src=twsrc%5Etfw

[15] https://t.co/g0QPauqlZz

[16] https://twitter.com/skeleton_claw/status/980844960189566976?ref_src=twsrc%5Etfw

> Our future is either that we're all Uber drivers and take turns driving each other around, or we're all podcast hosts and take turns being each other's guests…I just can't decide which one is worse □
>
> — Ashley Mayer (@ashleymayer) April 6, 2018[17]

We reached the point in which we do not believe anymore when things just work. Not even MacBooks are reliable these days.

> Everything works. That's very strange. Something must be wrong. This cannot be.
>
> — fabrice □□□□□ (@fabricetdc) February 20, 2018[18]

What kind of world have we created? And yes, I say we, because we are the makers of this world. Did we want to create this?

What is this uneasy feeling we have in the back of our hearts and our minds, continuously? We know that the next gizmo or programming language will only bring trouble and more issues. We roll our eyes when a coworker touts a new framework. We sigh in frustration when we have to reboot our computers for the upteenth time to be able to do our jobs, because some layer in the seemingly infinite abstraction layer of our software stack has decided to become comfortably numb.

> Software engineering's history—a bunch of kids thrashing about in Javascript, until a small collection of academics solve their underlying problems, which are then unified by the fearsome intellect of Ada Lovelace—works best in reverse.
>
> — The Dogmatic Developer (@iwasleeg) February 21, 2018[19]

You most probably have heard by now that we are entering a new "cold war." But this one has been happening in computer screens for a while now. There are hordes of programmers like us disrupting the power grid and the hospitals of other countries. There are programmers like us writing software that drives missiles and Tomahawks and planes to kill people on the other side of the planet. There are programmers like us driving counterespionage strikes against other countries because the only logic we know is "tit for tat." There are programmers making apps and websites supporting government propaganda, or supporting corporations trying to manipulate elections and to keep the statu quo from ever changing.

We are those programmers. We are shaping the state of things.

> Ces années formidables early 80ies avec Amiga, C64, les premiers MacIntosh, les jeux et de la literature dédié 😁😂□

[17] https://twitter.com/ashleymayer/status/982117084682371072?ref_src=twsrc%5Etfw

[18] https://twitter.com/fabricetdc/status/965998612697288705?ref_src=twsrc%5Etfw

[19] https://twitter.com/iwasleeg/status/966308558399852544?ref_src=twsrc%5Etfw

> cC @Trogambouille[20] pic.twitter.com/0WXFQmEk1Q[21]
>
> — Sylvain Gardel (@sylvain_gardel) April 13, 2018[22]

And yes, to answer my retorical question above, this is the world we always wanted to build. We wanted this. Since always.

> A 1920s prediction of the horror and inconvenience that would occur if anyone ever invented a pocket telephone… pic.twitter.com/RrqhGRVUhl[23]
>
> — Myko Clelland (@DapperHistorian) January 31, 2018[24]

2. On Hypocrisy

A couple of years ago I stopped using Google. I remember that my friend Maximiliano Firtman[25] called me a "tech vegan". This was in 2013, around 5 years ago. I slowed down using Facebook and Instagram starting in 2014. This year I started slowing down in my use of Twitter.

In the meatime, I have also decided never to rent a car using Uber, or an holiday spot with AirBnB, and I evangelize around me to raise awareness about the issues they generate. I do not want to support systems that are actively undermining the life of workers, of communities, of citizens. I refuse to live in a world where my driver does not have enough to live and send their kids to school and have food and a roof to sleep. I refuse to be hosted in locations whose landlords refuse to rent them to locals, shortening the offer and raising the prices, while at the same time buying properties just to be available through online platforms.

In the same vein, I stopped buying e-books from Amazon; I do the inverse of what most people do. I check the book reviews on Amazon and then I go to the website of the author or the publisher itself, so that I can get a DRM-free book in EPUB format, that I can read in any reader, without being locked into the system of a company that is undermining the value of the work of writers like me. Many authors and publishers sell their books directly; this is the virtual equivalent of a next-door grocery store.

I want people to hire me as a developer instead of somebody else on the the other side of the planet. Of course I do. Similarly, I want the owner of my next door grocery store to thrive. I want everybody to be able to find a home to rent at a reasonable price. I want my

[20] https://twitter.com/Trogambouille?ref_src=twsrc%5Etfw

[21] https://t.co/0WXFQmEk1Q

[22] https://twitter.com/sylvain_gardel/status/984860531784847361?ref_src=twsrc%5Etfw

[23] https://t.co/RrqhGRVUhl

[24] https://twitter.com/DapperHistorian/status/958770207908130817?ref_src=twsrc%5Etfw

[25] http://firt.mobi/

local bookstore owner to be able to pay salaries to their employees, so that they can all pay the rent, and maybe even enjoy two weeks of holidays somewhere else under the sun.

Is that too much to ask?

We are drowned in this race to the bottom, being told that we need more productivity, that in the 24 hours of our day we must do 1h30 of gym, work 12 hours, commute 2 hours, eat healthy, get the kids from school, enjoy being stuck in jams, listen to audiobooks and podcasts, read 6 books a year, learn a new programming language, attend conferences and local user groups, put our kids to bed, spend time with our significant other, pursue our dreams, and sleep at least 8 hours.

Also, in the meantime I discovered DuckDuckGo, and actually it offers great search results, and as a result I have not used Google in over 3 years now. And I rediscovered Firefox, the browser that nobody uses anymore, and it actually works great and it has fantastic developer tools and it syncs great between iOS, Android, Windows, Mac and Linux, and that is exactly what I needed.

I also wrote most of my books in Vim, but that you knew already.

> pic.twitter.com/XRLYb4XWYk[26]
>
> — DJ Crizzy Pepz (@DJCrizzyPepz) August 3, 2017[27]

In general, I have to say that I became a big fan of the terminal. I do as much as I can with it. It sounds incredible, isn't it? Writing the future of technology using what arguably is the oldest available interface in the world of computing: the command line.

Why am I doing this? Why is it that I chose such an old medium to express the ideas of the future? Why am I choosing those tools instead of the next shiny gizmo?

> GCC was first released in 1987, and got comprehensible error messages in 2013. React was first released in 2013.
>
> — The Dogmatic Developer (@iwasleeg) October 2, 2017[28]

The reason is honesty. After 20 years of using every single version of Windows since 3.1 to 10, and every single version of macOS since Jaguar to High Sierra, I have to admit that I grew tired of the broken promises[29]. I grew tired of the crashes, the UI changes, the inherent instability.

The command line soothes me. I feel safer, living a simple life. It is like living in a farm, growing your own food, and being able to exchange those vegetables and those eggs with other like-minded farmers.

[26] https://t.co/XRLYb4XWYk

[27] https://twitter.com/DJCrizzyPepz/status/893144893476016128?ref_src=twsrc%5Etfw

[28] https://twitter.com/iwasleeg/status/914848287655710720?ref_src=twsrc%5Etfw

[29] /blog/courage/

Now, let me be clear on this point: command line apps crash as often as their GUI counterparts. But they do not pretend not to. They have absolutely no marketing around them. They just do one or two things, they do it quite good, and they get out of the way fast. They launch in nanoseconds, they are ready to use as soon as you boot your machine, and they disappear into oblivion fast. And they even eat less battery and CPU power than their GUI counterparts.

Paraphrasing the Agile Manifesto authors, **I came to value more honesty over hipocrisy**. I am tired of hipocrite tools, promising things they can never deliver. I am tired of software not working, full of security bugs, full of spyware and broken promises.

I do not want no more broken promises. I do not want no more spying on me. I do not want poor people in my communities. I do not want Silicon Valley spreading its shit on us anymore.

The interesting thing is that, for some reason, many people pay attention to my technology choices. I am an unspoken influencer. And guess what, you are too, unspoken influencers of your own surroundings. You have the power to change things.

That is why I know that in the next few days some of you will think about these words, you are going to remember them, and you are going to click the "download" button in some website to download some "unconventional" tool, and you will discover that in our great world of software, we can break free from vendors and their broken promises.

And I mean, all vendors. Yes, including Apple.

> I wish Apple would take whatever money they're putting into Xcode and just give it to Jetbrains.
>
> — Logan Johnson (@loganj) May 20, 2017[30]

3. On Helping Each Other

Last year I started offering coaching sessions over Skype. I feel honored and humbled by those who actually trusted me, a complete stranger, to listen to their issues. We have met for an hour and a half, over Skype, and we talked. They told me their issues, they told me their anguishes, their fears. I tried to give some perspective, some opinion, some ideas. I do not know if my recommendations worked or not, if they feel better or not. I hope.

I wanted to thank those people who have asked me for help. Yes, I want to thank you, because through you I have learnt how to deal with my own issues, how to cope with my own hysterical thoughts, and how to calm them down. I have learnt to listen, or at least I started to understand how listening works. You have made me a better person. You know who you are.

[30] https://twitter.com/loganj/status/866035719856414721?ref_src=twsrc%5Etfw

And in average, seriously, the state of our profession is quite depressing. Most of the people who contacted me are around 30 years old, and they have deep unanswered questions: "What do I do with my life? I have been writing iOS apps for 5 years," they tell me, "and I do not know what to do next." I also get a lot of "I am burning out at my work, I do not know how to talk to my boss[31], I feel I'm nobody in my company[32]..." "My company is doing well and I am making lots of money, but I know that they are doing shady business and I feel uneasy about it." "I feel like my job has absolutely no sense, and would like to open a hotdog shop in an island in Polynesia before global warming drowns it."

There used to be a life crisis moment at 40, remember? Well, I can say that, according to my observations and witnessing the stories of the people who asked me for help, the crisis now happens at 30.

Thirty years old and in crisis.

Can you imagine that? Well, of course you can, because the average age in this room is, precisely, 30 years old.

> The most observant among you will already have noticed that the chocolate anomaly has struck again. #nobel2017[33] pic.twitter.com/d9PKR6JVb1[34]
>
> — Prof. Take that, Chemistry!, PhD (@TakeThatChem) October 4, 2017[35]

Even here in Switzerland, one of the countries touted as the happiest in the world, we have people committing suicide and, sadly, among them, many talented software developers, some of which I had the immense privilege of being friends or working with.

They just decided to leave.

They decided to end their lives.

Why?

> Programming to me is a highly emotional act. My code is an extension of me.
>
> — Stephanie Hurlburt (@sehurlburt) July 23, 2017[36]

Why are we building this world? Is this the world we have built? Is this our reality? Why is it that we have to keep coping up with harassed groups of human beings? Women and other genders than men who cannot be paid at the same level of their male counterparts? Black developers actively pushed away from the industry? An industry made

[31] /blog/developers-and-politics/

[32] /blog/developers-and-politics/

[33] https://twitter.com/hashtag/nobel2017?src=hash&ref_src=twsrc%5Etfw

[34] https://t.co/d9PKR6JVb1

[35] https://twitter.com/TakeThatChem/status/915525066540093440?ref_src=twsrc%5Etfw

[36] https://twitter.com/sehurlburt/status/889112181635072000?ref_src=twsrc%5Etfw

of white men between 25 and 35 years old who could not give less of a shit about accessibility and inclusion?

> Long story short: I do not accept the argument that tech is so well paid it can't organize. Tech is so well paid IT SHOULD ORGANIZE.
>
> — Thomas H. Ptacek (@tqbf) April 27, 2017[37]

4. On Responsibility

Bill Gates built an empire and made himself the richest man on Earth around a single, simple idea:

The EULA. The "End User License Agreement" in which it is clearly explained that you cannot sue the company if you use Windows to run your weapon system or your nuclear plant. Actually, as a consumer you do not get much leverage at all, but we accepted it all in order to be a part of the new world order built by the failed hippies of the Bay Area who ended up working in technology companies.

> We now live in an age where the streams of content we choose to consume literally effect the way we perceive reality.
>
> — Andrew (@_McGibbon) November 18, 2017[38]

We wanted the shiny toys. We wanted to be a part of. We were suffering a strong case of FOMA and we accepted someone becoming billionaire without any accountability.

Without. Any. Accountability.

But the times are changing, and the good old EULA looks terribly misplaced today.

Hence Open Source appeared, and in the past 15 years it grew in popularity to the point where big surveillance systems such as Android are 100% open source. Of course you cannot expect your pull request to be merged anytime soon, because open source only means cheap workers to these companies, protected with a very vertical and established decision system shaped as a pyramid.

> MY SISTER TEACHES CELLO
>
> SHE REPORTS THAT HER NEW STUDENTS CALL SHARPS "HASHTAGS"
>
> CONCERTO IN F HASHTAG
>
> — Marian Call (@mariancall) September 6, 2017[39]

[37] https://twitter.com/tqbf/status/857656007320842241?ref_src=twsrc%5Etfw

[38] https://twitter.com/_McGibbon/status/931976677684629504?ref_src=twsrc%5Etfw

[39] https://twitter.com/mariancall/status/905485243620114432?ref_src=twsrc%5Etfw

This is exactly what happened back in the 70's in those communist experiments, hippies living together and taking huge amounts of LSD. They all ended up being very conservative, with women taking care of kids and having charismatic leaders at the top of those experiments.

The hippie movement was not a change; it was a shift. The underlying problem of society, that of the distribution of power, did not disappear.

Quoting Mike Monteiro, I wholeheartedly agree with his idea of regulating the software and design industries. Yes, I stand for the idea that we need to hold corporations responsible for the disasters going on in the planet. Twitter and Facebook have to be more active in blocking hate speech. Microsoft has to be more active in actually making sure that Windows is not used in life-support systems or nuclear plants (guess what: it is.)

On the other hand, it is tantamount for politicians to be accountable and proficient in technology issues. The time in which politicians and journalists laughed and bragged about their misconceptions and lack of knowledge of technology and technical issues is gone. We need MBA degrees to include technology as a subject (guess what, it is not part of the curricula.) We must make universities accountable for the lack of technical education of lawyers, doctors, sociologists, mechanical engineers, and even dentists. And, at the same time, we must make computer science students learn about ethics and philosophy.

> "It's not enough to ask if code executes correctly. We also need to ask if it makes society better or worse." https://t.co/OdYVdT4Y1M
>
> — Rachel Thomas (@math_rachel) October 3, 2017[40]

We need both fields, computer science and liberal arts to intertwine and become one.

Because, as a matter of fact, Computer Science is the new Liberal Arts.

Maybe Apple is in the intersection of both; I just hope that they will use their position with responsibility, that is all.

Computers are so ingrained in our society, that our programs shape behavior, they shape policies, they shape the taxes that we will have to pay, they shape the education of the next generation, and the colors of the next Autumn-Winter 2019 collection. Everything is shaped by software. Our world runs in software.

As software engineers, we are today the most important driving force in the planet. We have a huge responsibility, and this responsibility is much, much bigger than choosing between `npm` or `gulp` or `Carthage` or `Cocoapods`. Our responsibility is to make ethical choices.

[40] https://twitter.com/math_rachel/status/915296134733602816?ref_src=twsrc%5Etfw

Yet, as most of you know, there is absolutely no “Ethics” chapter in common Computer Science curricula. It is then, our responsibility to learn about it, just like you would a new programming language, and then live by it.

> Computer Science: know the risks. ☺ pic.twitter.com/nEwNMqspfq[41]
>
> — Mark Heckler □□□□✈ (@mkheck) September 3, 2017[42]

5. On Money

I regularly get some funny looks from otherwise nice people when I mention that I do not do Bitcoin.

I actually do not even do stocks. I do not have anything about that. I do not aim to be insanely rich. And apparently we are more and more every day thiking like this on in this planet. I am happy I can make a living, I live a very good life. I am healthy and I have food in my plate every day.

People are all like, what, don’t you wanna have a Lamborghini? Oh man I would buy that house overlooking Lake Lugano! Don’t you want to attend parties in Monte Carlo?

The answer is no. Seriously, I do not want that. And I do know that as soon as some of those people get insanely rich, they are going to actively try to destroy communities, they will actively work to undermine ecology, all because they want more.

More, more, more.

I want people to live a decent life. All people. Not just a few. And do not get me started about charities, because that is also a big part of what is going wrong in the world. I volunteered as a webmaster for a human rights NGO in Geneva for a few years, and what I saw was disgusting. The amounts of money spent in lobbying and travel and dinners and theft by the NGOs of the world is staggering. Now everybody is talking about the sex abuse scandals in those organisations, but seriously we should all take a closer look and realise that charity is **not** a solution.

If you want to make a difference, you do not need to join a charity and go to some place in Africa or Latin America to help. You can start in your neighborhood, buying from the local stores; instead of starting a company, starting a cooperative; going to the fresh food market; buying only what you need; recycling all that you waste; being a voluntary firefighter; contributing a pull request instead of writing a bug; writing a book or publishing a podcast calling to action and denouncing injustice; writing to your elected officials and joining protests outside your parlament; un-voting those officials after their term was over, and removing them from power if they become dictators.

[41] https://t.co/nEwNMqspfq

[42] https://twitter.com/mkheck/status/904417957740142592?ref_src=twsrc%5Etfw

Remember this: **every penny that you have, is a penny that somebody else does not have**. The idea is that we can all eat, not just the rich. The idea is that we can all go to school, not just the rich. The idea is that we can all find a job, not just the rich. The idea is that we can all live a decent life with good healthcare and maybe even enjoy a nice holiday with our family every year, not just the rich. The idea is that we all live in peace, not just the rich.

> The better, stronger, faster, harder life is bullshit. This is an amazing piece. https://t.co/aMZxh146Fs
>
> — Joshua Topolsky (@joshuatopolsky) December 18, 2017[43]

And guess what? The global GDP of the planet is enough for every human being to live a decent life. Each one of us can have enough food, water, education, healthcare, and peace, and in a world built out of software, it is our duty to make this happen.

Instead, we are doing exactly the opposite. Stop this bullshit of increasing stakeholder value. Stop this bullshit of working 60h per week. Stop this bullshit of stock options and IPOs. Stop this bullshit of cryptocurrencies. Stop this bullshit of running towards money. Open your eyes and learn to help each other, I beg you.

By helping a single human being you are helping the whole of mankind at the same time.

> There's a whole separate rant here about how companies want loyalty from employees, but aren't prepared to offer it in return.
>
> — Sarah Mei (@sarahmei) October 26, 2017[44]

6. On Our Real Customers

I was talking about all these things to Hernún[45], one of my best friends, and he told me something interesting. He's a software developer, by the way, he works with PHP, WordPress, and PhpStorm.

He told me this interesting anecdote; a company contacted him, a software agency, to make an automated management system for some software development task, and he listened to his customer telling him how this product would help them get rid of developers in the future, and then asked my friend how much did he think building this project would take.

At that point my friend politely refused the project, asking this simple question:

[43] https://twitter.com/joshuatopolsky/status/942775805549142018?ref_src=twsrc%5Etfw

[44] https://twitter.com/sarahmei/status/923425400448942080?ref_src=twsrc%5Etfw

[45] http://hernun.com.ar/

> Why would I work in a project that will ultimately help you get rid of me or others like me?
>
> Have we not everything that life can give In having one another?
>
> — W. B. Yeats (@Yeats_Quotes) December 21, 2017[46]

Who are we writing code for? Who are our customers? Why do we write software? For whom? On behalf of whom?

The truth is, we software developers we are **not** making the world a better place. At all. Because we write software for "increasing stakeholder value" instead of writing software for society as a whole.

Our software increases income and social inequality.

Our software helps governments to spy on us.

Our software kills people.

Our software is used to manipulate elections.

Our software segregates social groups.

I have written software for government agencies. I have worked for big, big, big multinational companies emptying the resources of the third world. I have made systems that have actually left people without a job after they were put into production.

For what? For whom?

Ask yourself those questions. Then answer them.

We are not merely software developers. We are society developers. With each line of code, we shape our world.

> "The future depends on giving people power again. Not power over. But power to. Power to realize themselves. The best, truest, and noblest in themselves." – @umairh[47]
>
> ❤ https://t.co/YprgRlZ0Ap
>
> — Aral Balkan (@aral) April 9, 2018[48]

7. On Technology As A Force For Good

My dear friend Adam Jones, one of the most brilliant software developers I have ever met, told me this the other day:

> Technology's biggest problems are society's biggest problems. The endless need for growth and more money. All else follows.

[46] https://twitter.com/Yeats_Quotes/status/943937400132243456?ref_src=twsrc%5Etfw

[47] https://twitter.com/umairh?ref_src=twsrc%5Etfw

[48] https://twitter.com/aral/status/983308858989989888?ref_src=twsrc%5Etfw

We have built the future that we wanted. Yet, as it turns out, the world now goes so fast that it is possible for a single generation, like mine, to vividly remember black & white TVs, long play disks, cassettes, CDs, Zip drives, iPods and AirPods. We went from watching Star Trek characters communicate with a handheld computer to actually have a handheld computer in our pockets for over a decade now. Over a decade, can you believe it?

> Heard Wind of Change during dinner tonight and it reminded me of a time when we tore down walls.
>
> — Dan Allen (@mojavelinux) April 15, 2018[49]

This means two things:

- Overload. Too much is going on. A lot.
- Also, the real possibility to change things and to witness that change in our lifetime.

Do you see what I mean? This is a realization: this is the first time in the history of Mankind in which we can witness the changes that we bring to society as a whole in a few years, months or even days. And I do not mean at the level of your neighborhood or city, but at the level of a species, at the level of a planet.

We have a REPL (Read, Evaluate & Print Loop) in our hands, one that has the size of a planet.

And with it we can actively change the world in so many, deep, ways, that you will be able see the effects of those changes as your kids will grow. You will see them thinking differently; not just using different gadgets, but actually building a different world. Their value system will shift. Their perception of power will be transformed. Their priorities won't be yours.

Chase[50] from Páraic Mc Gloughlin[51] on Vimeo[52].

Maybe the Age of Aquarius is finally here? Maybe we can stop thinking of the Agile Manifesto and simply start caring about our colleagues at work? Maybe we can start to use the advances in technology to actually work less, to spend more time with our families, to make more art, instead of increasing shareholder value? Could we maybe stop increasing and concentrating quotas of power in small groups of privileged yet completely unethical individuals? Can we maybe acknowledge the dynamics of power, so that we break them, instead of empowering them? Instead of building democracies like Ponzi Schemes where the top has it all and the bottom gets to vote and should feel happy about it, could we start building network democracies, where people can actually bring down those elected officials before their

[49] https://twitter.com/mojavelinux/status/985420885514317825?ref_src=twsrc%5Etfw

[50] https://vimeo.com/229457692

[51] https://vimeo.com/paraicmcgloughlin

[52] https://vimeo.com

term without them claiming “witch hunts” and while the press empties our minds and fills them with hatred?

We must build a world in which every worker can send their kids to school. A world where AI is used so that we can all live a bit better, not fearing that we are all going to die or to be homeless or hopeless or workless. Where everybody can find a place to rent, without anyone driving the prices up. Where software is not made to spy on us. Where we take the sane choice of growing our own food, consuming local, removing power from intermediates and feeding all the planet with sustainable food.

We, software developers, we have the power to change the world but so far we have been unworthy of that power. We have served the interests of big capital. We are building systems that are making this world a worse place. We are not helping.

> I just rebooted a light switch. What a time to be alive.
>
> — mikeash (@mikeash) April 14, 2018[53]

It is time to change. It is time for Google & Facebook employees to leave the company in masse, and to join cooperatives working for the greater good. It is time for you to stop using Uber, AirBnB and similar companies. It is time to stop using Chrome (and to stop making websites that only work on one browser or another, for the sake of our integrity it’s 2018 people!) Do not accept jobs where you are going to be asked to spy on people. Do not accept jobs where you are going to be asked to destroy the social tissue of a community.

It is time to change.[54]

There is people on the other side of Visual Studio Code and Xcode and AppCode and whatnot. People. People who eat, who die, who cry, who learn, who laugh, who teach, who suffer. Dealing with the daily struggle of bringing food to the table. The unbearable lightness of being. The uncomfortable truth of the tears of their kids. The unfulfilled need for a hug. The never happening moment of a smile. The breakpoint never being hit.

A better world is one where we all care for each other. Continuously, unashamedly, purposedly, passionately.

So stop at that breakpoint. Now. Think about that new world. Then write it. Test it. Deploy it. With empathy.

This talk is dedicated to the memory of my friend Bertrand.

Thank you so much for your attention.

> When you understand the outer then you can go very deeply inwardly and that inward depth has no limit

[53] https://twitter.com/mikeash/status/985180855571279872?ref_src=twsrc%5Etfw

[54] /blog/developers-learn-to-say-no/

— Krishnamurti (@K__Quotes) June 21, 2017[55]

Special thanks to Hernún, Adam Jones, Beatrice Sigrist Charbonnier, Jonathan Rothwell, and to the Twitter users still making it worthwhile.

[55] https://twitter.com/K__Quotes/status/877357930558799874?ref_src=twsrc%5Etfw

PJSIP Snippets

Adrian Kosmaczewski

2018-04-24

I've been working in an IP telephony project with PJSIP[1], and had to implement a few features here and there. Here's a couple of snippets I used all over the place.

Extract header value

```
pj_str_t event_hdr_name = pj_str("Event");
pjsip_generic_string_hdr *event_hdr = (pjsip_generic_string_hdr*)pjsip_msg_find
if (event_hdr == NULL)
    return NULL;
pj_str_t event_value = event_hdr->vhalue;
```

Source[2].

Add header to list

```
struct pjsip_generic_string_hdr header;
pj_str_t name = pj_str("Name");
pj_str_t value = pj_str("value");
pjsip_generic_string_hdr_init2(& header, &name, &value);

pj_list_push_back(&cfg.reg_hdr_list, &header);
```

Source[3].

Clone header

```
// Set the "Contact" header
pj_str_t contact_hdr_name = pj_str("");
pj_str_t contact_hdr_value = pj_str("");
const pj_str_t *name = pj_cstr(&contact_hdr_name, "Contact");
const pj_str_t *value = pj_cstr(&contact_hdr_value, "<sip:alice@192.168.7.12;tr
pjsip_generic_string_hdr *hdr = pjsip_generic_string_hdr_create(pool, name, val
```

[1] https://www.pjsip.org/
[2] https://stackoverflow.com/a/32959283/133764
[3] https://stackoverflow.com/a/33931875/133764

```
// Clone the "Contact" header
pjsip_contact_hdr *contact_hdr = (pjsip_contact_hdr*)pjsip_msg_find_hdr_by_name
pjsip_contact_hdr *clone_hdr = (pjsip_contact_hdr*)pjsip_hdr_clone(pool, contac
```

Add User-Agent header

```
pj_str_t ua_hdr_name = pj_str("User-Agent");
pj_str_t ua_hdr_value = pj_str("CSTAEngine");
status = csta_tdata_add_header(pool, *tdata, &ua_hdr_name, &ua_hdr_value);
```

Get origin IP address

```
// Requires a memory pool
pjsip_response_addr res_addr;
pjsip_get_response_addr(pool, rdata, &res_addr);
```

Build response to a server request outside of dialog

Keep a reference to the data:

```
static pjsip_rx_data *last_data_received;

// ...later, on the module callback functions...
pjsip_rx_data_clone(rdata, 0, &last_data_received);
```

Use it to create a response outside of a dialog:

```
// Respond to the original message
pjsip_endpt_respond(endpoint, NULL, last_data_received, PJSIP_SC_OK, NULL, &hdr
```

Minimal PJSIP Stack

Custom thread worker:

```
static pj_bool_t quit_flag = PJ_FALSE;
static int worker_thread(void *arg)
{
    PJ_UNUSED_ARG(arg);

    while (!quit_flag) {
        pj_time_val timeout = {0, 500};
        pjsip_endpt_handle_events(endpoint, &timeout);
    }

    return 0;
}
```

Custom stack setup:

```
pj_init();
pj_caching_pool_init(&cp, &pj_pool_factory_default_policy, 0);
```

```
pj_sockaddr addr;
pj_status_t status;

pj_log_set_level(3);

status = pjlib_util_init();
csta_log_assert(status);

status = pjsip_endpt_create(&cp.factory, NULL, &endpoint);
csta_log_assert(status);

pj_uint16_t sip_af = pj_AF_INET();

pj_sockaddr_init(sip_af, &addr, NULL, sip_port);
if (sip_af == pj_AF_INET()) {
    if (sip_tcp) {
        status = pjsip_tcp_transport_start(endpoint, &addr.ipv4, 1, NULL);
    }
    else {
        status = pjsip_udp_transport_start(endpoint, &addr.ipv4, NULL, 1, NULL)
    }
}
else if (sip_af == pj_AF_INET6()) {
    status = pjsip_udp_transport_start6(endpoint, &addr.ipv6, NULL, 1, NULL);
}
else {
    status = PJ_EAFNOTSUP;
}

csta_log_assert(status);

status = pjsip_tsx_layer_init_module(endpoint);
csta_log_assert(status);

status = pjsip_ua_init_module(endpoint, NULL);
csta_log_assert(status);

//        pj_bzero(&inv_cb, sizeof(inv_cb));
//        inv_cb.on_state_changed = &call_on_state_changed;
//        inv_cb.on_new_session = &call_on_forked;
//        inv_cb.on_media_update = &call_on_media_update;
//        inv_cb.on_rx_offer = &call_on_rx_offer;

//        status = pjsip_inv_usage_init(endpoint, &inv_cb);
//        csta_log_assert(status);

status = pjsip_100rel_init_module(endpoint);
csta_log_assert(status);
```

```
pj_pool_t *pool = pjsip_endpt_create_pool(endpoint, "", 1000, 1000);
status = pj_thread_create(pool, "", &worker_thread, NULL, 0, 0, &thread);
csta_log_assert(status);
```

Remember to add a module to be notified of events:

```
pj_status_t status = pjsip_endpt_register_module(endpoint, &csta_module);
```

Minimal PJSUA Stack

Much simpler than using bare bones PJSIP:

```
pj_status_t status;

pj_caching_pool_init(&cp, &pj_pool_factory_default_policy, 0);

status = pjsua_create();
csta_log_assert(status);

pjsua_config cfg;
pjsua_config_default (&cfg);
NSString *userAgentString = [PJSUALibraryManager userAgentString];
cfg.user_agent = pj_str((char*)[userAgentString cStringUsingEncoding:NSUTF8Stri

pjsua_logging_config log_cfg;
pjsua_logging_config_default(&log_cfg);
log_cfg.console_level = 4;
log_cfg.decor &= ~(PJ_LOG_HAS_TIME | PJ_LOG_HAS_MICRO_SEC);

// Init the pjsua
status = pjsua_init(&cfg, &log_cfg, NULL);
csta_log_assert(status);

pjsua_transport_config cfg2;
pjsua_transport_config_default(&cfg2);
cfg2.port = 0;

// Add UDP transport.
status = pjsua_transport_create(PJSIP_TRANSPORT_UDP, &cfg2, NULL);
csta_log_assert(status);

pjsua_transport_config cfg3;
pjsua_transport_config_default(&cfg3);

// Add TCP transport.
status = pjsua_transport_create(PJSIP_TRANSPORT_TCP, &cfg3, NULL);
csta_log_assert(status);

// Initialization is done, now start pjsua
status = pjsua_start();
csta_log_assert(status);
```

Enseñando C++

Adrian Kosmaczewski

2018-07-17

Una vez se me ocurrió enseñarle C++ a un amigo. Un error garrafal, pero bueno, fundamentalmente, lo que le dije fue esto.

1. La memoria asignada a un proceso está dividida en pila ("stack"), montón ("heap") y "static store". Esto es así desde que Von Neumann inventó la computadora.
2. Cada vez que ves un par de { } en el código, es un nuevo "stack frame" o "contexto" que se crea en la pila. Todas las variables locales van ahí. Todas.
3. A diferencia de muchos otros lenguajes, C++ permite crear objetos en la pila:

```
string s { "test" };
vector<int> v {1, 2, 3};
```

Estas dos instrucciones crean objetos en la pila. Estos objetos **desaparecen** cuando llega el } de cierre del contexto o stack frame actual.

4. Pero también se pueden crear objetos en el montón, y para que no se pierdan, les asignamos un "puntero" en la pila:

```
string *s = new string { "test" };
vector<int> *v = new vector<int>  { 1, 2, 3 };
```

El string y el vector están en el montón; en la pila tenemos dos punteros (que básicamente contienen la dirección en memoria de dónde están los objetos en si).

5. Si pierdes los punteros, tenés un "memory leak" ya que **es imposible** recuperar la posición original de los objetos en el montón.
6. Si usas la palabra clave `static` tu objeto termina en el... static store.
7. Obviamente los corchetes de inicialización de los objetos no son los mismos corchetes que los que se usan para delimitar stack frames.
8. Lo importante es que asimiles las dos sintaxis de creación ya que tienen semánticas diferentes.

9. El montón se usa para objetos que viven larga vida o que son enormes; el stack es perfecto para objetos pequeños, temporarios y para valores escalares como enteros o números de coma flotante.
10. Cada vez que usas “new” sos el dueño del objeto y por lo tanto es obligatorio que uses “delete” cuando no necesitas más el objeto.
11. Java, PHP, C# usan un “recolector de basura” que escanea cada tanto el montón y borra los objetos que ya no tienen dueño. C++ no tiene eso.
12. C++ tiene una librería estándar impresionante pero por razones históricas y/o técnicas muchas librerías vienen con sus propios string, array, etc, y hay que aprender a usarlos, a sabiendas de que en otro proyecto, quizás, uses otro Array u otro string. Es lo que hay.

Mas cosas interesantes aca: http://cppquiz.org

De Programmatica Ipsum

Adrian Kosmaczewski

2018-10-01

De Programmatica Ipsum is a new monthly treatise on individuals, interactions, and the true valuation of the things on the left, to be published on the first Monday of each month.

De Programmatica Ipsum[1] will be a monthly publication, the collaboration of Graham Lee[2] and myself, self described as “two old burnt out guys shouting into the `void *`.”

De Programmatica Ipsum will feature new articles on the first Monday of every month about the craft of software engineering, the life of software developers, with a focus in issues like inclusion, burnout, startup life, hype and heresy, and other related topics.

De Programmatica Ipsum won’t use advertising or analytics.

Please subscribe to our newsletter[3] to be notified of new releases as they are available, or contribute[4] if you would like to support our work.

[1] https://deprogrammaticaipsum.com/
[2] https://www.sicpers.info/about/
[3] https://deprogrammaticaipsum.com/newsletter/
[4] https://deprogrammaticaipsum.com/contribute/

12 Years of iPhone – A Developer's Perspective

Adrian Kosmaczewski

2019-04-21

This is the talk that I gave in the 4th MCE Conference[1] in Warsaw, Poland, on May 8th, 2017 (conference organized by Polidea[2]) and (with updates) at UIKonf[3] on May 15th, 2018 and at NSConfArg[4] on April 20th, 2019.

Introduction

The iPhone celebrates its 12th anniversary this year. From a historical point of view, it had a tremendous impact in the industry and the careers of those involved in mobile application software development.

At some point in my career, I was a .NET developer, and then one day I told myself that I wanted to write Objective-C for a living. This is how I started my career in this galaxy. This is how I got here. I could have chosen Windows Mobile. I could have chosen BlackBerry. I chose the iPhone.

I started working on my first iPhone application back in July 2008, and back then everybody, and I mean every single person in the industry around me, told me that I was completely crazy and/or stupid for losing my time with a device nobody would buy. I have not stopped writing iOS applications ever since, even though lately I've been spending quite a bit of time around Android.

The iPhone turned out to be a far, far bigger platform than any of us could ever imagine. In this talk I am going to take you in a trip back in time, to remember frameworks, people, companies, events and projects that have marked our craft in the past decade.

The memories of the iPhone will bring back the birth of the App Store, which was (and to a certain extent still is) one of the most controversial features of the platform; the iPad; the Apple Watch; the rise of the Apple TV; the Mac App Store; the visual changes; the jump from

[1] http://2017.mceconf.com/
[2] https://www.polidea.com/
[3] http://www.uikonf.com/
[4] http://nsconfarg.com/

Lucida Grande to Helvetica to San Francisco; and many other events, some funnier than others, that shaped this story. But more than anything, this story will run around the personal story of a man called Steve Jobs.

This is the story of the best-selling electronics consumer product in history, and how it shaped the mobile industry.

2007

On January 1st, AAPL closed at 12.25 USD[5].

I spent Christmas 2006 and New Year 2007, in Madrid, Spain. I got news of the release of the iPhone in the airport of Madrid, ready to go back home, on January 10th, 2007. I told my wife, right there, right then, that if Apple ever released a software development kit for that device, we would be both going to San Francisco to attend WWDC.

Steve Jobs introduced the iPhone in a now legendary sequence[7], stating that it was three products in one, a mobile phone, a touchscreen iPod, and a breakthrough internet communications device. The device was introduced to the public months before its official introduction to the public, to allow Apple to submit the required paperwork for the FCC, which consists of publicly disclosed documents.

At the end of the presentation, Alan Kay reportedly told Jobs[8]:

> "When the Mac first came out, Newsweek asked me what I (thought) of it. I said: Well, it's the first personal computer worth criticizing. So at the end of the presentation, Steve came up to me and said: Is the iPhone worth criticizing? And I said: Make the screen five inches by eight inches, and you'll rule the world."

After the presentation of the iPhone, James Duncan Davidson[9] took an iconic photo of the iPhone inside a glass screen[10].

On the other side of the fence, Steve Ballmer just laughed at the iPhone.

In May, Steve Jobs used the term "post-PC device"[11] during his interview with Bill Gates at the D5 conference:[12]

[5] The stock prices used in this article take into account the various stock splits[6] that Apple has made during its lifetime.

[7] https://www.apple.com/pr/library/2007/01/09Apple-Reinvents-the-Phone-with-iPhone.html

[8] https://gigaom.com/2010/01/26/alan-kay-with-the-tablet-apple-will-rule-the-world/

[9] https://twitter.com/duncan

[10] https://www.yatzer.com/duncan-davidson-beauty-detail/slideshow/6

[11] http://allthingsd.com/20070531/d5-gates-jobs-transcript/

[12] https://www.youtube.com/watch?v=ZWaX1g_2SSQ

> But then there's an explosion that's starting to happen in what you call post-PC devices, right? You can call the iPod one of them.

– Steve Jobs

What I found most interesting about the iPhone was that it ran a modified version of OS X – which meant that the programming toolkit had to be based on Objective-C, which was by far one of my preferred programming languages. I had started learning Objective-C back in 2002, after I bought an iBook with OS X Jaguar – my first ever version of OS X back then. I wanted to work as an Objective-C developer, I wanted to leave the Windows PC behind and use only Macs at work. This was clearly my chance.

WWDC 2007 happened, and although we all knew that Objective-C was somewhere down there, Steve Jobs famously stated that the only way to create iPhone applications would be using web technologies. We were all wrong. While it was true that Safari on the iPhone was absolutely groundbreaking, I was (and still am) more interested in creating native apps.

Joe Hewitt's[13] released the first iteration of the iUI[14] framework on July 5th 2007,[15] originally called "iphoneav." This was the first mobile web framework targeting the iPhone. Joe was already a superstar at the time, having released Firefox and Firebug[16] and for being later responsible of the early versions of the Facebook application.

The iUI framework was very important because it set the tone for many other similar frameworks that appeared later on, combining semantic HTML with special CSS effects and JavaScript code to create iPhone-like experiences as native applications; many other similar frameworks followed this pattern, such as jQTouch[17], WebApp.net[18] and later jQuery Mobile[19], Sencha Touch[20] or even lately Ionic[21].

The iPhone went on sale on June 29th. It was quite an underpowered machine by current standards: 128 MB RAM, 4 to 16 GB of storage, a 32-bit CPU made by Samsung, underclocked at 412 MHz. No front camera, a 2.0 MP rear camera, and it lacked many, many other things:

> … the iPhone didn't support 3G, it didn't support multitasking, it didn't support 3rd party apps, you couldn't copy or paste text, you couldn't attach arbitrary files to emails, it didn't support MMS, it didn't support Exchange push email,

[13] https://en.wikipedia.org/wiki/Joe_Hewitt_(programmer)
[14] http://www.iui-js.org/
[15] https://en.wikipedia.org/wiki/IUI_(software)
[16] https://web.archive.org/web/20091115094010/http://getfirebug.com/
[17] http://jqtjs.com/
[18] http://trywebapp.net/
[19] http://jquerymobile.com/
[20] https://www.sencha.com/products/touch/#overview
[21] http://ionicframework.com/

> it didn't have a customizable home screen, it didn't support tethering, it hid the filesystem from users, it didn't support editing Office documents, it didn't support voice dialing, and it was almost entirely locked down to hackers and developers.

– The Verge[22]

In July 2007 I opened an account in a young website started the previous year, called Twitter.

Justine Ezarik received a 300-page bill from AT&T in a box, and her video became legend. It was customary before smartphones for mobile network providers to include details of all messages, communications and expenses in their bills, but the amount of communications enabled by iPhone quickly changed the game – much too quickly for AT&T, as a matter of fact.

In a bit more than two months, Apple sold one million iPhones,[23] and Time named it the "invention of the year."[24]

The first "jailbreak" options appeared merely days after the iPhone went on sale; hackers all over the planet, like Erica Sadun, started to install their own native applications on rooted devices. Remember PwnageTool and JailbreakMe, Cydia, and even the first version of Twitterrific?

The success of the iPhone and of these early attempts to hack its core was such that Jobs changed his mind and finally acknowledged, in a press release in October 17th 2007[25], that Apple would be releasing an SDK for the iPhone the following year.

The waiting time had to do with polishing what would ultimately become the first version of UIKit, including the security features required for developers to sign their applications to enable their installation on any device. In his piece, Steve Jobs mentions the need for tight security in iPhones, given the possibility of viruses, and highlights how Nokia disabled arbitrary code execution in their devices for similar reasons.

The nature of the iPhone – a small, fast computer in your pocket, with access to contextual information about the user, including but not limited to contacts, preferences, location and other data, was a significant shift from traditional desktop platforms, and thereby Apple considered that it required a completely different approach.

[22] https://www.theverge.com/2011/12/13/2612736/ios-history-iphone-ipad

[23] https://www.apple.com/pr/library/2007/09/10Apple-Sells-One-Millionth-iPhone.html

[24] https://content.time.com/time/specials/2007/article/0,28804,1677329_1678542_1677891,00.html

[25] https://arstechnica.com/apple/2007/10/apple-to-open-iphone-ipod-touch-to-third-party-developers-in-early-2008/

Needless to say, after hearing the news, I reserved the tickets and the hotel for San Francisco right away.

In December, AAPL reached 28.30 USD.

In their quarterly statements of 2007, Apple declared having made 24B in revenue selling 52M iPods, 7M Macs and 1.1M iPhones. The iPhone was only available in the USA, UK, Germany and France.

2008

On January 1st, AAPL closed at 19.34 USD.

Apple introduced the first official iPhone SDK in March 6th. As important as the SDK, or even more, was the announcement by Steve Jobs of the approval process of the App Store. This was the first time that Apple announced a developer platform in which there would be a validation process, including a right to veto, in order to release applications. This model was inspired by the game console industry, in which console makers can veto games if required. This feature led to a huge outcry, most of which has still not faded, but it turned out to be one of the simplest solutions to the problem of malware in mobile devices.

To this day, thanks to the review process, in spite of all the vitriol that it still generates in the developer community, the iOS platform remains virtually free of malware and viruses.

In April the first “Iron Man” movie is released, kickstarting the Marvel Cinematic Universe franchise. In June followed the second film of the franchise, “Hulk.”

A code-sharing website called Github started operations in April, aiming to be a better and bigger version of Google Code and based around Git.

In July 2008 I attended my first ever WWDC, in San Francisco. The event sold out in 2 months, the first time this happened in the history of the WWDC.

During the keynote, Steve Jobs said that[26]

> “… in the first 95 days, 250,000 people downloaded the iPhone SDK. 25,000 developers applied, and 4,000 were admitted in the program.”

During the same keynote, Scott Forstall delivered the first of his customary demo sessions, with over 10 different companies showing what developers could do using the iPhone SDK. And also, Mobile Me happened, which was, well, not really ready for prime time.

In terms of software, the iPhone SDK lacked lots of things we take for granted today in the platform:

[26] https://www.cnet.com/news/live-blog-steve-jobs-at-wwdc-2008/

- Interface Builder was a separate application from Xcode 3.
- No Instruments.
- No ARC: `retain`, `release` and `autorelease` were our friends.
- No IBOutletCollection.
- No Storyboards.
- No Auto Layout.
- No Gesture recognizers.
- No Map views.
- No Unit testing (OCUnit / SenTest did not work with iPhone apps back then.)
- No Nitro JavaScript engine.
- No Pull-to-refresh.

But it had one serious bug: a memory leak in `[UIImage imageNamed:]` which drove everybody crazy at some point.

The Barenaked Ladies played in the WWDC Bash, for those who remember.

The iPhone 3G was released to the public during WWDC, July 11, 2008, and this time in many more countries, including Switzerland.

The App Store opened in June 2008 with only 544 apps; it turns out that there were 72 registered rejections.

My wife and I got our first iPhones back from San Francisco: 8 GB iPhone 3G, with pretty much the same specifications as the first iPhone, but with a 3G connectivity and with an integrated GPS. As Steve Jobs said during the WWDC keynote, location-based services was going to be one of the biggest breakthroughs of the new device.

Surely enough, my first iPhone application was a GPS-enabled social networking system, long gone from the App Store since then, but the one that started my career as an iPhone developer.

During those early days, somebody published the 1000 USD "I am Rich" application[27] – and actually sold a few copies, before removing it (or being removed?) from the App Store.

Of course the whole WWDC event was under a fucking NDA[28] until September; this was unfortunate, because it meant that nobody could answer questions about the iPhone SDK in this new website called "Stack Overflow," which started in September 15th.

Precisely on that day, September 15th, the collapse of the investment bank Lehman Brothers, the largest bankruptcy filing in U.S. history, with over $600 billion in assets, triggers the worst financial crisis in modern history. The bankruptcy triggered a drop in the Dow Jones index of 4.4%, followed by a another of 7% on September 29th.

In October I co-organised the first and last iPhone Conference in

[27] http://www.kottke.org/08/08/the-1000-iphone-app

[28] https://www.wired.com/2008/08/iphone-coders-feel-miffed-muzzled-by-apple-s-nda/

Geneva – we even got a cease-and-desist letter from Apple for using the word “iPhone” in the name of the event, and we ended up not having permission to use any pictures of iPhones to promote it.

On November 21st, the first edition of “Beginning iPhone Development” by Jeff LaMarche and Dave Mark[29] goes on sale and becomes an instant hit, the first published book explaining the iPhone SDK.

Groups of developers devoted to the iPhone SDK started appearing all over the planet. In French-speaking Switzerland we had our own, organised by a company called EasyBox from Lausanne. The “Groupe de développeurs iPhone de la Suisse Romande” will forever stay in our memories for almost three years of continuous meetings, and for enabling countless business ventures and anecdotes.

In their quarterly statements of 2008, Apple declared having made 32B in revenue selling 54M iPods, 9M Macs and 11.6M iPhones. Cumulated sales for the iPhone in 2008 amounted to 12.7M units. The iPhone made its biggest debut in more than 70 countries, including Switzerland in July and Poland and Argentina in August.

There were 13’000 apps available in the App Store at the end of the year.

2009

On January 1st, AAPL closed at 12.88 USD.

In January, Joe Hewitt (yes, him again) released Three20[30], arguably the most important early library of code for iPhone applications. It was wonderful in terms of functionality but quite heavy to integrate in its entirety. It is historically relevant in that it spawned an entire industry of components, fueled by Stack Overflow and Github.

Also in January, Palm introduced WebOS[31] as a contender in the touchscreen smartphone race. In June, the first smartphone using this operating system, the Palm Pre, is released in the USA.

In February, Brian Acton and Jan Koum, both former employees of Yahoo!, incorporate a company named WhatsApp.

That year I quit my job and started my first company, called “akosma software.” During this time I wrote iPhone and Android apps, trying to bootstrap my business. I released `nib2objc` a small command line tool that transformed XIB files into its equivalent Objective-C code, and Erica Sadun wrote an article about it in Ars Technica.[32]

[29] https://www.amazon.com/Beginning-iPhone-Development-Exploring-SDK/dp/1430216263/

[30] https://github.com/facebookarchive/three20/tree/cc672132abf4539e2fa61b2b9f382fb3e88dc49a

[31] https://en.m.wikipedia.org/wiki/WebOS

[32] https://arstechnica.com/apple/2009/04/iphone-dev-convert-xib-files-to-objective-c/

In April, PhoneGap, a small project created by a Canadian company named Nitobi, wins the People's Choice Award at O'Reilly Media's 2009 Web 2.0 Conference. PhoneGap was arguably the first cross-platform app development kit targeting iOS and Android, using the native web views of each platform.

WWDC 2009 was the second one for me in June that year. This time it sold out in just one month. During this event, Apple introduced iPhone OS 3 with many features; this version was more a "bug fix and missing features" edition than anything else, adding all the features that all analysts expected from a "serious" smartphone platform. Apple focused its attention on iOS, in detriment of OS X Snow Leopard, famously known to be a "no new features, just stabilization" release.

Among the new features for users:

- Clipboard support
- Turn by turn navigation
- Magnetometer
- Spotlight search
- Push notifications
- Horizontal keyboard
- MMS
- Voice control

And many new frameworks for us developers to play with:

- MapKit.framework
- CoreData.framework
- ExternalAccessory.framework
- GameKit.framework (including the first version of Bluetooth P2P functionality for data and audio)
- MessageUI.framework
- StoreKit.framework

During the event Apple unveiled the iPhone 3GS, the first hardware update since the first iPhone, and the first release to have the "S" (for "Speed") next to its name. Visually identical as the iPhone 3G, it featured a substantial improvement in hardware: 256 MB or RAM, a 600 MHz ARM CPU, a 3 MP rear camera with video capabilities, faster Bluetooth and Wi-Fi, and up to 32 MB of storage.

The iPhone 3GS was officially supported by Apple until February 21, 2014, when iOS 6.1.6 was released, almost 5 years later, and it was still manufactured and sold until September 2012.

The band Cake played during the WWDC Bash in the Yerba Buena Gardens. Phil Schiller hosted the keynote in replacement of Steve Jobs, who had taken a leave of absence at that time. Scott Forstall delighted the audience once again with a long parade of app developers demoing their applications.

I started teaching iPhone development in 2009, as the interest of developers in the platform was growing; of course, Objective-C and its

manual memory management turned off way too many developers accustomed to garbage collected runtimes.

In November 2009, Luke Wroblewski published his seminal article “Mobile First”[33] triggering a deep change in the thought process and the design of mobile applications and websites.

In December, Verizon published an ad for the new Droid phone, using a picture uncannily similar to the iconic one[34] taken by James Duncan Davidson[35] in 2007.

In their quarterly statements of 2009, Apple declared having made 36.5B in revenue selling 54M iPods, 10.3M Macs and 20.7M iPhones. Cumulated sales for the iPhone in 2009 amounted to 33.5M units. The iPhone was made available in 13 new countries during that year such as China, South Korea and Saudi Arabia.

There were more than 100’000 apps available in the App Store.

2010

On January 1st, AAPL closed at 27.44 USD.

On January 7th, NBC aired “Planet of the Apps”[36] – a documentary about the app economy, which by now was becoming seriously huge. Oddly enough, the name “Planet of the Apps” will be reused years later… by Apple themselves. But more on that later.

In January Steve Jobs introduced the iPad; it brought a larger screen to a public who thought of it as just a “larger iPod touch. The iPad found a niche as an”intermediate” kind of computer, one that is great for travel, kids and elderly users, and one that stands at the crossroads or rather at the border[37] between technology and liberal arts.

For developers, the release of the iPad brought iPhone OS 3.2, an interim release of the operating system with gesture recognizers and “Universal Apps,” that is, apps that showed a different UI for the iPhone or for the iPad, but using the same binary code for both. This reduced the cost of developing apps for this platform, but of course required revisiting the UI architecture of many applications to adapt to this new screen size.

In February Samsung introduced a new mobile platform, called Bada[38], which would be discontinued three years later.

[33] https://www.lukew.com/ff/entry.asp?933

[34] http://www.cultofmac.com/22718/spot-the-difference-iconic-photo-of-iphone-new-verizon-droid-tv-ad/

[35] https://twitter.com/duncan

[36] http://www.cnbc.com/id/34465283

[37] http://www.sicpers.info/2013/12/standing-at-the-crossroads/

[38] https://en.m.wikipedia.org/wiki/Bada

In April, Hewlett Packard bought Palm and announced plans to release smartphones and tablets using WebOS[39].

In April Steve Jobs wrote its famous "Thoughts on Flash"[40] piece, in which he explained the technical impossibility for Adobe to include the Flash plugin in mobile systems. History showed that not only he was right about iOS, but also about Android; in neither case Adobe made the Flash plugin a reality.

Apple counterattacked Adobe's, strategy of creating applications using Flash by adding clause 3.3.1 to the iPhone SDK agreement, explicitly forbidding applications written in any other programming language[41] than Objective-C, C or C++ in the App Store.

After substantial backslash and controversy, which included prominent Mac developer Jonathan "Wolf" Rentzsch canceling his C4 developer conference[42], Apple removed said clause in September.[43]

In April, RIM (the company that made BlackBerry smartphones) bought the company who created the QNX real-time operating system[44], and announced new smartphone and tablet platforms called BlackBerry 10[45] and BlackBerry Tablet OS[46] based on it.

Also in April, the third movie of the Marvel Cinematic Universe was released: "Iron Man 2."

In June I went to the USA for my third WWDC in a row; this one sold out in just eight days. In this event, with a very skinny Steve Jobs on stage, iPhone OS became iOS 4 and all of a sudden this operating system became a workhorse for the first time, loaded thousands of new features, some of which never before seen on a mobile operating system.

For developers, iOS 4 was just like Christmas ahead of time:

- A brand-new Xcode 4, integrating Interface Builder into a single application – one that was slightly buggy and very unstable for a long while.
- ARC: Automatic Reference Counting, also known as "the compiler manages your memory;" one of the most important features in the platform today, was unveiled at this WWDC 2010 to a cheering crowd, who could barely believe what Apple was announcing.
- Objective-C blocks: this proposed extension to the C language, already available in OS X, and somewhat similar to C++ lamb-

[39] https://en.m.wikipedia.org/wiki/WebOS

[40] https://web.archive.org/web/20200430094807/https://www.apple.com/hotnews/thoughts-on-flash/

[41] http://daringfireball.net/2010/04/iphone_agreement_bans_flash_compiler

[42] https://www.cnet.com/news/apple-iphone-move-kills-mac-coding-conference/

[43] https://www.apple.com/pr/library/2010/09/09Statement-by-Apple-on-App-Store-Review-Guidelines.html

[44] https://en.m.wikipedia.org/wiki/QNX

[45] https://en.m.wikipedia.org/wiki/BlackBerry_10

[46] https://en.m.wikipedia.org/wiki/BlackBerry_Tablet_OS

das[47], sparked the birth of new functional APIs all over the Cocoa frameworks.
- Building upon blocks, Grand Central Dispatch[48] (also known as `libdispatch`) was also introduced that year, and brought a new simpler pattern for writing concurrent software.
- Core Text: this new framework, almost imported verbatim from OS X, allowed developers to display text on their applications like never before.
- Multitasking: starting with iOS 4, developers of music, location or voice-over-IP applications could have them running on the background for real. This brought interesting new challenges and possibilities, such as the capability to perform long-running download operations on the background, without fearing the application to be killed in the meantime.
- Support for higher density displays: this was the birth of the @2x hack for high-resolution images.
- The new iAd network, allowing developers to include Apple-approved advertising in their applications, which would last until 2016.

New frameworks available for developers:

- CoreMotion.framework
- AssetsLibrary.framework
- CoreTelephony.framework
- CoreText.framework
- EventKit.framework
- EventKitUI.framework
- MobileCoreServices.framework
- QuickLook.framework
- iAd.framework

Of course, iOS 4 brought really cool new features for users:

- FaceTime.
- App folders.
- Retina display.
- Support for iPad.
- Game Center.

And some changes related to multitasking:

- Task completion.
- Local notifications.
- App switching.

In the same event, Apple introduced the new iPhone 4, sporting a design that would be the iconic “iPhone look” until 2014. This new device was seriously capable, and included the brand-new Retina dis-

[47] https://mikeash.com/pyblog/friday-qa-2011-06-03-objective-c-blocks-vs-c0x-lambdas-fight.html

[48] https://en.wikipedia.org/wiki/Grand_Central_Dispatch

play, which made pixels disappear from sight forever. The iPhone 4 was my second iPhone.

It had an Apple A4 SOC with a 32-bit ARM CPU at 1 GHz, 512 MB RAM, up to 32 GB of storage, and for the first time, a 0.3 MP front camera capable of video at VGA resolution. The back camera was a 5 MP one, capable of HD video at 30 fps. This device was supported until iOS 7.1.2, released June 30, 2014.

Right after the release of the iPhone 4, Antennagate happened. Steve Jobs had to come out and publicly defend its innovative design in the face of reception problems, triggered or not depending on how users held the device in their hands.

In July, a photographer named Lee Morris did a full fashion photo shoot with an iPhone 3GS.[49] "iPhoneography" was already a thing.

On October 6, 2010, a new iOS app named Instagram was released on the App Store. Also in October Apple registered a trademark for the phrase "There's an app for that."[50]

Some really popular open source libraries back then:

- ASIHTTPRequest[51]
- JSONKit[52]
- InAppSettingsKit[53]

And in other news, the word "App" was declared word of the year[54], althought Steve Jobs used it already in NeXT promotional videos back as early as in 1989.

Microsoft released by the end of the year a new mobile platform competing with Android and iOS, called "Windows Phone."

Android started to seriously take off, and overtook the iPhone in absolute number of activated devices by the end of the year, as Google famously announced 300'000 device activations per day[55].

In their quarterly statements of 2010, Apple declared having made 65.2B in revenue selling 50M iPods, 13.6M Macs, 40M iPhones and 7.5M iPads. Cumulated sales for the iPhone in 2010 amounted to 73.5M units. The iPhone reached 3 new countries during that year, Vietnam, Armenia and Tunisia.

There were more than 300'000 apps available in the App Store.

[49] https://fstoppers.com/commercial/iphone-fashion-shoot-lee-morris-6173

[50] http://edition.cnn.com/2010/TECH/mobile/10/12/app.for.that/index.html

[51] https://allseeing-i.com/ASIHTTPRequest

[52] https://github.com/johnezang/JSONKit

[53] http://www.inappsettingskit.com/

[54] http://www.americandialect.org/app-voted-2010-word-of-the-year-by-the-american-dialect-society-updated

[55] http://appleinsider.com/articles/10/12/09/google_activating_300000_android_devices_per_day_for_free

2011

On January 1st, AAPL closed at 48.47 USD.

On January 6th, Apple released the Mac App Store to the public, taking inspiration from iOS. The sandboxing requirements for apps released through the Mac App Store blocked many app makers, particularly those producing developer tools, from selling their apps in it.

Mike Lee, a well-known Mac and iOS software developer who had worked at Apple and Delicious Monster, started Appsterdam, an initiative to bring together the best app developers under the idea that "if you want to make movies, go to Hollywood; if you want to make apps, come to Appsterdam."

An obscure patent holder company called Lodsys launched trolling patent claims against many well known developers[56] selling apps in the App Store, such as James Thompson[57] or the The Iconfactory[58]. These claims had a strong impact in the platform, increasing the uncertainty around the App Store. This whole affair faded into obscurity after a few months, and to this day it is unknown what was the solution Apple found to settle these claims.

In April a company named JetBrains released a new IDE for Objective-C, named AppCode[59], providing for the first time an alternative to Xcode, and putting a refactoring menu never before seen in the hands of iOS developers.

Also in April, the 4th movie of the Marvel Cinematic Universe is released: "Thor." In July comes the 5th: "Captain America: The First Avenger."

In May two entrepreneurs named Wayne Chang and Jeff Seibert started Crashlytics[60]. More or less at the same time, Miguel de Icaza, the founder of the Mono project[61], announced the creation of a new company called Xamarin[62].

In June Flickr announced that the iPhone 4 was the most popular camera[63] used in the site.

WWDC 2011 sold out in just 12 hours. I did not attend that one; Apple introduced iOS 5 to developers, with many new features:

- The new Notification Center
- iMessages
- Newsstand

[56] https://www.theguardian.com/technology/2011/may/13/apple-iphone-developers-app
[57] http://pcalc.com/
[58] http://iconfactory.com/
[59] http://www.jetbrains.com/objc/
[60] https://get.fabric.io/
[61] http://www.mono-project.com/
[62] https://dotnet.microsoft.com/apps/xamarin
[63] https://www.engadget.com/2011/06/21/iphone-4-most-popular-camera-on-flickr/

- Reminders
- Twitter integration
- Music application (formerly the “iPod” application)
- Users can use the volume button to take pictures
- Rich text formatting in mail messages
- New basic photo editing features in Photos
- Safari Reader mode
- Siri (only on the iPhone 4S)
- iCloud (replacing Mobile Me)

New frameworks available to developers in iOS 5:

- Accounts.framework
- CoreBluetooth.framework
- CoreImage.framework
- GLKit.framework
- NewsstandKit.framework
- Twitter.framework

The most important (and controversial) feature of iOS 5 for developers was, without any doubt, the introduction of Storyboards. This extension of the classic XIB format included the possibility to define navigation and composition patterns in a visual manner, something never before seen in the platform.

iAd started to lower its minimum amount of advertising contract to $500,000 in February, to $300,000 in July, apparently to bring back some big advertisers who had left the platform.

In August, Hewlett Packard stopped all smartphone and tablet efforts, effectively killing the WebOS[64] platform.

The iPhone 4S was released in October, and it was the first iOS device that did not require tethering to a Mac or PC during setup. It featured a dual core 32-bit Apple A5 SOC, at 1 GHz, with 512 MB of RAM, an 8 MB rear camera capable of HD video at 1080p, and a 0.3 front camera with VGA resolution. For the first time, a 64 GB option for storage was available. It also put an end to the Antennagate controversy. The iPhone 4S was my third iPhone.

Eloy Durán released the first public version of CocoaPods[65] in September 1st and the world of iOS software development changed forever; later on, Orta Therox would take over its development.

In other news, after appearing in the WWDC keynote, Steve Jobs retreated from public life and passed away in October 5th, a week before the release of iOS 5 and the iPhone 4S. His official biography, written by Walter Isaacson, was released shortly thereafter to great controversy.

In October, Adobe puchased Nitobi, the company behind the PhoneGap framework. The code of PhoneGap was donated to the Apache

[64] https://en.m.wikipedia.org/wiki/WebOS
[65] https://cocoapods.org/

project, and became Apache Cordova.[66]

In their quarterly statements of 2011, Apple declared having made 108.3B in revenue selling 42M iPods, 16.7M Macs, 72M iPhones and 32M iPads. Cumulated sales for the iPhone in 2011 amounted to 146M units. Cumulated sales for the iPad in 2011 amounted to 40M units. All in all, Apple sold more than 185M iPhones and iPads since 2007. The iPhone became available in Bolivia, Slovenia and Trinidad and Tobago.

There were more than 500'000 apps available in the App Store.

2012

On January 1st, AAPL closed at 65.21 USD.

In January Objective-C ranked as "language of the year" in the TIOBE Index.[67] The rise in popularity of the language is arguably due to the interest sparked by the iOS platform.

In February Apple introduced the first iPad with Retina screen, and lowered the price of the minimum contract for iAd to $100,000.

Also in February, Firefox presents the Firefox OS[68] smartphone platform.

In March, Facebook bought Instagram for USD 1 billion.

In April the 6th film of the Marvel Cinematic Universe is released: "The Avengers." It had to be renamed in the UK to avoid confusion with those other Avengers.[69]

In May, Google buyed Motorola Mobility[70] for USD 12.5 billion.

In June I went to my fourth WWDC in San Francisco, which would be my last. It sold out in one hour and 43 minutes. During the Keynote, the first without Steve Jobs on stage, Scott Forstall showed the new features of iOS 6, Apple Maps and Passbook, as well as a tighter integration with Twitter, Facebook and other social networks. Other features brought by iOS 6 were some enhancements for Siri, Shared Photo Streams, iCloud Tabs for Safari and the fact that FaceTime calls could be done through cellular networks at last.

There were other minor enhancements all over the platform, but by far the biggest and most controversial change in iOS 6 is the change from Google Maps to Apple Maps, one that unfortunately had been rushed out; at the time of this writing Apple Maps is still behind Google Maps in features and quality, and its many problems led to major outcry on the press and on social media.

[66] https://cordova.apache.org/
[67] https://www.tiobe.com/tiobe-index/
[68] https://en.m.wikipedia.org/wiki/Firefox_OS
[69] https://en.wikipedia.org/wiki/The_Avengers_%28TV_series%29
[70] http://money.cnn.com/2012/05/22/technology/google-motorola/index.htm

In all likelihood, the debacle caused by Apple Maps was enough to drive Scott Forstall out of the door, and to consolidate the image of Apple as a faulty provider of cloud services, one that started with iTools, .Mac and Mobile Me, and one that sticks to the company five years later[71].

iOS 6 brought a few new frameworks for developers to play with, but it was not a ground-breaking release in terms of changes or additions:

- AdSupport.framework
- MediaToolbox.framework
- PassKit.framework
- Social.framework

The most important new element in the developer toolkit was the introduction of Auto Layout and the `NSLayoutConstraint` class, which found its full raison-d'être a few months later, when the new iPhone featured a new screen size, and user interfaces would have to automatically support the new dimensions while keeping their proportions and usability.

The iPhone 5 was released in October, and it was the first iOS device that had a screen size bigger than the classic 320x480 points that the iPhone supported since its inception. It featured a dual core 32-bit Apple A6 SOC, at 1.3 GHz, with 1 GB of RAM (first device with such an amount,) an 8 MB rear camera capable of HD video at 1080p, and a 1.2 MP front camera with HD video capabilities. Available in 16, 32 or 64 GB options of storage. It was the first iPhone that supported LTE or "4G" networks.

It is the oldest device still supporting iOS 10.3.1 at the time of this talk, but Apple has already announced that iOS 10.3.2 will not support 32-bit devices any more.

In September, AAPL reached an all-time high of USD 95.30.

In October, after more than a year in preparation, Mattt Thompson[72] released version 1.0 of AFNetworking[73], certainly one of the most popular iOS open source libraries ever released in the platform.

In October, Apple released the iPad mini, the first model with a screen of 8 by 5 inches, somehow fulfilling Alan Kay wishes of a tablet "worth criticizing."[74]

That same month, the first MacBook Pro with Retina screen became available, and now the @2x hack for images was also available in the Mac.

[71] While reviewing this article, Graham Lee made this observation: "An interesting pattern is that almost everything people complain about from Apple (iCloud, iTunes, the App Store) is from divisions that report to Eddy Cue."

[72] http://nshipster.com/authors/mattt-thompson/

[73] https://github.com/AFNetworking/AFNetworking

[74] https://gigaom.com/2010/01/26/alan-kay-with-the-tablet-apple-will-rule-the-world/

The iPhone 5 and the iPad mini, as well as the late-2012 Retina iPad are the first iOS devices to support the new “Lightning” connector, which replaced the old, venerable 30-pin Apple Dock connector, introduced in 2003 on the 3rd generation iPod. This connector change prompted the whole market to buy new connectors, bumpers, adaptors and accessories of all kinds.

In their quarterly statements of 2012, Apple declared having made 156.5B in revenue selling 35M iPods, 18M Macs, 125M iPhones and 58M iPads. Cumulated sales for the iPhone in 2012 amounted to 271M units. Cumulated sales for the iPad in 2012 amounted to 98M units. All in all, Apple sold more than 369M iPhones and iPads since 2007.

There were more than 700’000 apps available in the App Store at the end of the year.

2013

On January 1st, AAPL closed at 65.07 USD.

Also in January, Objective-C ranked as “language of the year” in the TIOBE Index[75] for the second year in a row. More or less at the same time, Twitter announced the acquisition of Crashlytics for USD 100 million, Twitter’s largest purchase at the time.

Since the release of iOS 6 the previous year, the word “Skeuomorphism” invaded the press and social media. The word refers to a design trend to reproduce the look and feel, including textures and shadows, of physical objects in a computer screen, in this case an iOS touchscreen.

The time had come for the first major redesign of iOS in almost 7 years, coupled to the ousting of Scott Forstall and the rise of Jonathan Ive as head of the iOS design team.

In February Samsung shut off the Bada[76] platform, merging it into the Tizen[77] project, and Xamarin released Xamarin Studio[78], a set of IDE and frameworks geared towards the creation of cross-platform iOS and Android applications using the C# programming language.

The fate of BlackBerry looked very dim. During the year, Jim Balsillie[79] and Mike Lazaridis[80], executives from RIM, both sold a large amount of their shares. In January the company launched the BlackBerry 10 operating system, with 70’000 apps available in its store, but this could not help the company reverse its fortunes.

[75] https://www.tiobe.com/tiobe-index/

[76] https://en.m.wikipedia.org/wiki/Bada

[77] https://en.m.wikipedia.org/wiki/Tizen

[78] https://dotnet.microsoft.com/apps/xamarin

[79] http://business.financialpost.com/fp-tech-desk/rim-co-founder-jim-balsillie-has-sold-all-his-shares-in-blackberry

[80] http://www.huffingtonpost.ca/2013/12/24/mike-lazaridis-sells-blackberry-shares_n_4499461.html

In April the 7th film of the Marvel Cinematic Universe was released: “Iron Man 3.” In October came in the 8th: “Thor: The Dark World.”

In May Crashlytics added Android support.

WWDC 2013 sold out in just one minute and eleven seconds. iOS 7 was the most important announcement, its biggest release since its inception. A major overhaul in the whole platform, including a complete redesign of the user interface, bringing a whole new idiom and new capabilities never before seen in the platform.

For users, iOS 7 brought a redesign of the notification center, new icons, a flattened design (not always popular), the Control Center for quick access to the flashlight, AirDrop, a new Photos app, iTunes Radio, and a new card-based multitasker. It also brought FaceTime Audio, which turned out to be quite popular but also frowned upon by mobile network operators.

From a developer point of view, what stood out of iOS 7 was that applications could be updated automatically; this has had a dramatic change in the way people consumed applications, and in the way developers released them.

There was a large array of new frameworks available for developers in iOS 7:

- GameController.framework
- JavaScriptCore.framework
- MultipeerConnectivity.framework
- SafariServices.framework
- SpriteKit.framework

All of these including new classes and features:

- The new `NSURLSession` family of classes
- Support for iBeacons
- The new `[NSData base64EncodedStringWithOptions:]` method,
- The new `@import` syntax for importing modules or frameworks
- The `CMStepCounter` class for counting steps
- The `MKGeodesicPolyline` class
- The default hiding of MAC addresses by the operating system for privacy purposes
- The support for reading QR codes through `AVMetadataObject-TypeQRCode`
- Smile detection through `CIFaceFeature`
- Text-to-speech through `AVSpeechUtterance`
- The new Text Kit built upon Core Text
- Dynamic Type

Last but not least, the system font in iOS changed from Helvetica to Helvetica Neue.

iAd lowered its minimum contract to $50 in June. The writing was on the wall.

In August, the first movie about Steve Jobs was released, starring Ashton Kutcher in the lead role. Also in August, a company called Tiny Speck, founded by a certain Stewart Butterfield, starts an internal chat system called Slack. The name stands for "Searchable Log of All Conversation and Knowledge."

Apple introduced two successors to the iPhone crown: first the fast, not-coloured iPhone 5S, featuring for the first time a 64-bit dual core CPU, an Apple A7 SoC coupled with the new Apple M7 motion processor chip. With 1 GB of RAM and storage options up to 64 GB, including for the first time the Touch ID fingerprint sensor, nowadays standard all over the iOS device line, and a step counter sensor. The iPhone 5S was my fourth iPhone.

But I did not get the Gold version. Always Space Black for me, please.

The second successor to the iPhone 5 was the 5C, basically the same model than the 5 but cheaper, and sold in different colors, available in 8, 16 and 32 GB options of storage, with a 32-bit CPU at 1 GHz.

Later that year Apple introduced the first iPad Air, a lighter design for the device which became the standard iPad look and feel still in use today in the Pro version.

By December, 78% of all iOS devices had iOS 7 installed.

In their quarterly statements of 2013, Apple declared having made 171B in revenue selling 26M iPods, 16.5M Macs, 150M iPhones and 71M iPads. Cumulated sales for the iPhone in 2013 amounted to 421M units. Cumulated sales for the iPad in 2013 amounted to 169M units. All in all, Apple sold more than 590M iPhones and iPads since 2007. The iPhone became available in Sri Lanka.

There were more than a million apps in the App Store.

2014

On January 1st, AAPL closed at 71.51 USD.

At the beginning of the year, a rather serious vulnerability was found in iOS 6, still running a substantially large portion of the iOS codebase. This is the famous `goto fail;` error, causing an error during the SSL handshake, and leading to the possibility of snooping communications from and to iOS devices.

In January, Google sold Motorola Mobility[81] to Lenovo for USD 2.91 billion, two years after having bought it for USD 12.9 billion. The company calls this operation a "success."

In February Apple bought the popular beta-testing platform TestFlight[82] and later turned it into a relatively closed system. It removed

[81] https://www.theverge.com/2014/1/29/5358620/lenovo-reportedly-buying-motorola-mobility-from-google

[82] https://www.macrumors.com/2014/02/21/apple-may-acquire-testflight/

Android support, removed support for earlier versions of iOS, applications have to be approved by Apple, there was only limited support from Apple, and only the latest build was available for download. Developers do not care, they are using Crashlytics almost exclusively.

Also in February, Facebook bought WhatsApp for USD 19.3 billion.

In March was released the 9th movie in the Marvel Cinematic Universe: Captain America, The Winter Soldier. In July came the 10th: Guardians of the Galaxy.

Apple started the "lottery ticket system" for WWDC tickets. From now on, developers have to enrol for the opportunity to pay for a ticket, and I applied twice since then, without luck. I guess they know I was there four times before that.

The major announcement at WWDC 2014 was, without any doubt, the release of the Swift programming language[83] to the world. The brainchild of Chris Lattner, the creator of LLVM[84], Swift was presented as the natural heir and successor of the venerable Objective-C. With much stronger typing, featuring advanced features like generics, lambdas and type inference, the language became an instant hit and a major new driving force to bring new developers to the platform.

Another distinctive feature of the language are the much, much longer compile times, in comparison to Objective-C, naturally relaxed in this aspect.

iOS 8 was a major update, touted as the biggest ever of iOS. iCloud Drive became a real file sharing option, for both the Mac and iOS, and HealthKit and HomeKit opened the door to integrate iOS in new contexts.

There were many new features for developers:

- AVKit.framework
- Accelerate.framework
- AudioToolbox.framework
- CloudKit.framework
- CoreAudioKit.framework
- CoreAuthentication.framework
- HealthKit.framework
- HomeKit.framework
- LocalAuthentication.framework
- Metal.framework
- NetworkExtension.framework
- NotificationCenter.framework
- Photos.framework
- PhotosUI.framework
- PushKit.framework
- SceneKit.framework

[83] https://swift.org/
[84] http://llvm.org/

- WebKit.framework

iOS 8 can be thought of as the “Extensions” version of iOS, for many different services: third-party keyboards, data sharing, and others; Xcode had all the corresponding templates at hand. Even the Touch ID system was opened for third-party developers!

In July, Apple[85] and IBM[86] signed an unprecedented partnership to develop enterprise applications around iOS. This partnership, still in work today, has led to the deployment of iOS devices in countless companies around the world.

Such new features required a new version of the iPhone, and Apple did not disappoint; the release of the new iPhone 6 and iPhone 6 Plus, both featuring bigger screens (4.7 and 5.5 inches respectively) than their predecessors, were instant hits and proved the usefulness of the Auto Layout capabilities introduced in iOS 6.

They both included an Apple A8 SoC with dual core 64-bit CPUs at 1.4 GHz, up to 128 GB of storage (for the first time ever,) 1 GB of RAM, an 8 MP rear camera capable of 1080 HD video and a 1.2 MP front camera.

During the same presentation, Tim Cook introduced Apple Pay.

However popular as they were, they suffered the “Bendgate” by which users complained that, if left in pockets while they sit on them, the iPhones would bend. Which sounds quite logical when you say it out loud but it made people angry nevertheless.

In September Apple introduced the Apple Watch, opening the door to software developers to create applications for this new device; Apple Watch applications were initially “extensions” built-in inside iOS applications and running in the iPhone itself, but they became later fully-fledged applications in their own right. The Watch would be released the following year.

It took 4 weeks to iOS 8 to reach 50% of all iOS devices; by December it had reached 65%.

In October Steve Ballmer confessed to Charlie Rose his regrets not to have paid more attention to the phone business.

Very silently, Hewlett Packard shuts off the last services related to WebOS[87] in October.

In November, version 0.1 of the Carthage project[88] was published, becoming the first serious alternative to CocoaPods.

[85] https://www.apple.com/pr/library/2014/07/15Apple-and-IBM-Forge-Global-Partnership-to-Transform-Enterprise-Mobility.html

[86] http://www-03.ibm.com/press/us/en/pressrelease/44370.wss

[87] https://venturebeat.com/2014/10/15/goodbye-webos-hp-will-discontinue-services-for-the-last-remaining-devices-running-the-os/

[88] https://github.com/Carthage/Carthage

In their quarterly statements of 2014, Apple declared having made 183B in revenue selling 14M iPods, 19M Macs, 169M iPhones and 67M iPads. Cumulated sales for the iPhone in 2014 amounted to 590M units. Cumulated sales for the iPad in 2014 amounted to 237M units. All in all, Apple sold more than 827M iPhones and iPads since 2007. The iPhone became available in Serbia and Kosovo.

There were more than 1.3M apps in the App Store.

2015

On January 1st, AAPL closed at 117.16 USD.

The developer world started this new year chanting the carol of Swift, which would become the most loved programming language in the Stack Overflow survey at the end of the year.

Conferences, books, blog posts, podcasts, everybody in the developer community started praising the language, and promptly started rewriting the world in it.

In March a new biography of Steve Jobs was released, "Becoming Steve Jobs"[89] with the support from Tim Cook and other high ranking executives.

In April, the 11th film of the Marvel Cinematic Universe franchise was released: Avengers: Age of Ultron. In June came the 12th, Ant Man.

During WWDC 2015, Chris Lattner showed the second version of the Swift programming language, announcing that it would be made open source by the end of the year. iOS 9 and watchOS 2 are also announced, including enhancements to Siri, Apple Music and other "me too" features inspired from Android and Windows.

All the platforms produced by Apple now feature the same font, San Francisco.[90]

In June Apple released the "Move to iOS" Android application to help users switch their data to the iPhone.

The Apple Watch went on sale, and made Apple the 2nd biggest watch maker in the planet[91] in just a few months.

The new iPhone 6S and 6S Plus bring new standards to the platform: Apple A9 SoC with dual-core 64-bit CPUs at 1.85 GHz, with 2 GB of RAM (for the first time,) storage options up to 128 GB and a 12 MP rear camera capable of 4K video recording. They also feature 3D Touch, the possibility of triggering actions on applications by hard-pressing on the screen. The usefulness of this last feature is still debatable at best.

[89] https://en.wikipedia.org/wiki/Becoming_Steve_Jobs:_The_Evolution_of_a_Reckless_Upstart_into_a_Visionary_Leader

[90] https://medium.com/@mach/the-secret-of-san-francisco-fonts-4b5295d9a745

[91] https://www.wareable.com/apple/watch-sales-rolex-tim-cook-556

The iPad Pro was the first major upgrade to the iPad lineup since 2012, and was brought together with the Apple Pencil and the Smart Keyboard, bringing the interaction level of the iPad on par to that of similar laptops, and using a design and concept that was similar to that of the Microsoft Surface introduced a few years earlier. The iPad Pro featured up to 128 GB of storage, 4 GB of RAM, an Apple A9X Soc with dual-core 64-bit CPU at 2.26 GHz and an 8 MP rear camera. These specs were unprecedented in the platform so far.

A new Apple TV also came out at the end of 2015, featuring an App Store for applications built exclusively for it. Apple TV applications are distributed as part of standard iOS applications, increasing the perceived value of the platform to consumers. This new Apple TV comes bundled with a remote control that seems designed with hatred and which triggers automatically every time you sit on it.

A film adaptation by Aaron Sorkin and directed by Danny Boyle of the biography of Steve Jobs, with Michael Fassbender starring in the title role, was released on October 9th.

Xcode 7 came out and featured a migration assistant to port all of your Swift 1 code to Swift 2. You can call it a migraine assistant if you want.

In October, David Heinemeier Hansson, creator of Ruby on Rails, published a piece about the App Store model, called “Don’t base your business on a paid app.”[92]

The first releases of the server-side frameworks Perfect, Vapor, Zewo and Kitura are released, showing that there is a market interest in server-side solutions written in Swift. Kitura is produced by IBM as part of their partnership with Apple, and it shows the interest of IBM to use Swift as the next Java. The release of Swift for Linux as part of the open source project makes them immediately available for running in computers running the Linux operating system, even on Raspberry Pi devices.

By December, iOS 9 was installed in 75% of all active iOS devices.

In their quarterly statements of 2015, Apple declared having made 234B in revenue selling 20.5M Macs, 231M iPhones and 54M iPads. Cumulated sales for the iPhone in 2015 amounted to 821M units. Cumulated sales for the iPad in 2015 amounted to 291M units. All in all, Apple sold more than 1.11B iPhones and iPads since 2007.

There were more than 1.8M apps in the App Store.

2016

On January 1st, AAPL closed at 97.34 USD.

[92] https://m.signalvnoise.com/don-t-base-your-business-on-a-paid-app-a6440f33dd4c

In January, Apple announced that the iAd network was going to shut down in June.

In February WhatsApp announced over one billion users, and Microsoft bought Xamarin for a price estimated between 400 and 500 million USD.

In May, Instagram changed its icon.

In March Apple released the iPhone SE as a successor of the 5S, and a new 9.7-inch big iPad Pro as well, including a 4K-capable 12 MP rear camera, and a storage option of up to 256 GB, for the first time in the platform.

Apple celebrated its 40th anniversary in April 1st. Also in April, the 13th film of the Marvel Cinematic Universe was released: Captain America: Civil War. In October came the 14th: Doctor Strange.

Swift 3, Xcode 8 and iOS 10 were all shown in June during WWDC 2016, and released in September together with the new iPhone 7.

For users, iOS 10 allowed for the first time to hide the icons of the stock applications installed by Apple; it had new bubble effects in the iMessage application; a new Home application (fundamental complement for the HomeKit framework;) a new design for Apple Music; Apple Pay support in Safari; a Magnifier; and collaboration features in the Notes application.

For developers, iOS 10 brought new possibilities:

- Customize and interact with notifications
- The possibility to create iMessage apps to be sold through the App Store
- A new Siri framework
- Tighter integration of custom content in Maps
- The new CallKit framework for VoIP applications to integrate with the phone application
- A new speech recognition engine: `SFSpeechRecognizer`
- The True Tone display that dims the screen following the current lightning
- Better search through Core Spotlight and the `CSSearchQuery` class
- `UIPasteboard`, available since iPhone OS 3, now also works for copy-pasting between devices
- New app extensions: Call Directory, Intents, Intents UI, Messages, Notification Content, Notification Service, and Sticker Pack.
- New Photo capture API: `AVCapturePhotoOutput`

Apple Pay was available in many new countries, including Switzerland starting in July.

Also in July Apple announced the sale of the Billionth iPhone[93], mak-

[93] https://www.apple.com/newsroom/2016/07/apple-celebrates-one-billion-iphones.

ing it the most successful product in modern history[94] all categories combined:

> The following is a list of the best-selling products across several categories:
>
> Car model: VW Beetle 21.5 million
> Car brand: Toyota Corolla 43 million
> Music Album: Thriller 70 million
> Vehicle: Honda Super Cub 87 million
> Book Title: Lord of the Rings 150 million
> Toy: Rubik's Cube 350 million
> Game console: Playstation 382 million
> Book series: Harry Potter Series 450 million
> Mobile Phone: iPhone 1 billion
>
> The iPhone is not only the best selling mobile phone but also the best selling music player, the best selling camera, the best selling video screen and the best selling computer of all time.
>
> It is, quite simply, the best selling product of all time.

– Horace Dediu.

The iPhone 7 and iPhone 7 Plus were released in September, with the following specifications: Apple A10 Fusion SoC with quad-core 64-bit CPUs at 2.34 GHz, 2 or 3 GB of RAM, up to 256 GB of storage, a 12 MP rear camera capable of 4K recording and a 7 MP front camera. Of course the most important non-feature of the iPhone 7 was the lack of a headphone jack, which is proof of courage.

Apple released an updated, waterproof version of the Apple Watch.

In September all development in Firefox OS[95] ceases completely.

It took two weeks for iOS 10 to reach 50% of active devices. By December it was installed in 75% of them.

In their quarterly statements of 2016, Apple declared having made 215B in revenue selling 18.5M Macs, 212M iPhones and 45.5M iPads. Cumulated sales for the iPhone in 2016 amounted to 1.03B units. Cumulated sales for the iPad in 2016 amounted to 337.5M units. All in all, Apple sold more than 1.3B iPhones and iPads since 2007.

There were more than 2.1M apps in the App Store at the end of the year, more than a million of them compatible with the iPad.

2017

On January 1st, AAPL closed at 121.35 USD.

---

html

[94] http://www.asymco.com/2016/07/28/most-popular-product-of-all-time/

[95] https://en.m.wikipedia.org/wiki/Firefox_OS

I spent Christmas 2016 and New Year 2017 in Madrid, just as I did 10 years earlier. 1 billion iPhones earlier.

On January 10th, Chris Lattner, the creator of Swift and LLVM, left Apple and joined Tesla. Later the same month, Crashlytics announced that they were being purchased by Google.[96]

In March, Apple released the new iteration of the iPhone SE, and then changed the file system[97] of pretty much every iOS, tvOS and watchOS device in the planet (a number counted in billions) and nobody noticed. The new file system is focused on Solid-State Drives and encrypted volumes – that is, the present and the future. Another notable releases so far this year were the iPhone 7 Product RED and the AirPods.

The iPad Pro has today as much RAM as the first iPhone had of SSD storage: 4 GB.

Apple unveiled the trailer for "Planet of the Apps" – a reality show[98] with Jessica Alba, Gwyneth Paltrow and Will.i.am.

Analysts are starting to notice that sales of Apple devices have been slowing down for the past two years, even if there is still room to grow. But the headlines of financial magazines do not talk about the crisis of the independent software maker, suffering from the very structure of the App Store:

> If your goal is just to make money temporarily (which is up to you), then the race to the bottom — with all its attendant risks, and its environmentally corrosive effect — is probably your best bet. You also need to acknowledge that you've marked your work as being essentially worthless, and that it'll be discarded just as quickly. Your most vocal supporters will turn on you the minute you ask for more money (remember the extra levels for Monument Valley?). They simply won't value you enough to even consider paying again, because you've already taught them that your work isn't worth it.

– Matt Gemmell[99]

Apple started a great app crackdown, removing apps that were not available as 64-bit binaries, removing copycat apps and other utilities such as anti virus and spam. For the first time since its inception, the number of apps in the App Store actually decreased.[100]

[96] https://fabric.io/blog/fabric-joins-google

[97] https://en.wikipedia.org/wiki/Apple_File_System

[98] http://www.nbcbayarea.com/news/local/Planet-of-the-Apps-Developers-Have-60-Seconds-to-Pitch-Ideas-to-Celebrities-in-New-Apple-Show-413759073.html

[99] http://mattgemmell.com/damage/

[100] https://www.theverge.com/2018/4/5/17204074/apple-number-app-store-record-low-2017-developers-ios

WWDC 2017 happened in San Jose, instead of San Francisco for the first time since 2003. The lottery system is well in place right now.

Apple unveiled Swift 4 (but no Application Binary Interface,) Xcode 9 (which crashed less often,) iOS 11 (which crashed more often,) watchOS 4, tvOS 11 and macOS 10.13 “High Sierra” (which definitely crashed more often.)

iOS 11 featured the following new features:

- 64-bit only
- ARKit
- Drag & Drop
- CoreML
- Camera Depth API
- Core NFC
- Files.app

Later that year, Apple released three new iPhones, all of them water resistant for the first time:

- The iPhone 8 and 8 Plus;
- And the iPhone X (“ten”) which, just as Mac OS X (“ten”) got people saying “iPhone eks” all over the place. This was the first full-screen iPhone with the new “Face ID” technology.

Disney released the 15th to 17th films in the Marvel Cinematic Universe franchise: Guardians of the Galaxy, Volume 2, Spider Man: Homecoming and Thor: Ragnarok.

It took two months for iOS 11 to reach 50% of active devices. By December it was installed in 70% of them.

In their quarterly statements of 2017, Apple declared having made 229B in revenue selling 19.2M Macs, 217M iPhones and 43.7M iPads. Cumulated sales for the iPhone in 2016 amounted to 1.25B units. Cumulated sales for the iPad in 2016 amounted to 381M units. All in all, Apple sold more than 1.6B iPhones and iPads since 2007.

There were more than 2M apps in the App Store at the end of the year, half of them compatible with the iPad.

2018 And Beyond

On January 2nd, AAPL closed at 172.26 USD.

Today, the current version of iOS is 11.3.1. iOS 11 is installed in at least 76% of all devices. The oldest supported device at the time of this talk is the iPhone 5S, released in 2013.

This year the Marvel Cinematic Universe released Black Panther, Avengers: Infinity War and Ant Man and the Wasp, the 18th to 20th films of the highest-grossing film franchise of all times.

Swift 5, Xcode 10, iOS 12, macOS 10.14, watchOS 5, & tvOS 12 are all most probably in the horizon, although there will be no fixed Application Binary Interface this year, and WWDC will happen next month in San Jose once again.

To this day, Apple has sold 1.37B iPhones, 403M iPads, for a cumulated total of 1.8B iOS devices (not counting iPod touch, TV and WATCH devices!) Of course not all of these devices are currently in use. As I write this, AAPL hovers at 190 USD.

The iPhone is the most popular product of all time.[101] The most controversial. The most hated. The most loved. The sign of our times. There was a before and after the iPhone.[102]

But what comes next? It is our responsibility, as software developers, to pay attention to the impact that this little device has had in our culture, our politics, our education, our entertainment, and our life. We make the apps for these almost 2 billion devices. We have the responsibility of using that power with care, with ethics, and to ensure that not only we provide fast, performant, beautiful code to our customers; but also, that we actively seek to make this world a better one, if not for us, at least for the next generation.

Thank you so much for your attention.

[101] http://www.asymco.com/2016/07/28/most-popular-product-of-all-time/
[102] http://www.guerrilla-innovation.com/archives/2014/10/000862.php

The Greatest Unsolved Problem in the Digital Era

Adrian Kosmaczewski

2019-08-29

Thank you very much for inviting me to speak today. It is an honor and a privilege to be able to contribute to your event with some words that I hope, will trigger some positive thoughts and bring some new ideas to your brain.

About Myself

Just a few words about myself. My name is Adrian Kosmaczewski, I am a software developer currently working as a Developer Relations person and conference speaker.

In my 22 years of professional software engineering experience I wore many hats. Up until last year I worked for a decade as a mobile app developer, publishing apps for iOS and Android from 2008 to 2018. I was actually one of the first few iOS developers in Switzerland. Before that I was an ASP.NET developer, and even before that I was doing just plain ASP pages with VBScript, at a time when the word "Netscape" actually meant something.

I am also a writer, having published a certain amount of books about software engineering. Currently I am the co-author of De Programmatica Ipsum[1], a monthly magazine where I explore with my colleague Graham Lee the human side of software engineering.

Today

You have come from all over the world (quite literally, as a matter of fact) to think about the future of Digitalization. Today we talked a lot about things like the "Internet of Things," "Blockchain," "Disruption," "Machine Learning" and "Artificial Intelligence," "Big Data" and many other buzzwords.

Quite exciting, is not it? I am pretty sure you have plenty of ideas, suggestions, projects you would like to offer your customers – and also treats you want to offer yourselves! We in this industry we are

[1] https://deprogrammaticaipsum.com/

curious, always looking for new things to learn and new solutions to create.

Yet, before we close this event, I would like to talk about the context of those ideas. I would like to call your attention to the outer world, the one we are changing with our ideas. The one we do not always see as we write code, deploy our containers or upload our mobile applications.

The Purpose Of A Corporation

A few days ago, Graham Lee, my partner in crime at De Programmatica Ipsum, sent me this link from the "Business Roundtable," an "association of chief executive officers of America's leading companies working to promote a thriving U.S. economy and expanded opportunity for all Americans through sound public policy."

https://www.businessroundtable.org/business-roundtable-redefines-the-purpose-of-a-corporation-to-promote-an-economy-that-serves-all-americans

Here is a quote from the announcement:

> WASHINGTON – Business Roundtable today announced the release of a new Statement on the Purpose of a Corporation signed by 181 CEOs who commit to lead their companies for the benefit of all stakeholders – customers, employees, suppliers, communities and shareholders.
>
> Since 1978, Business Roundtable has periodically issued Principles of Corporate Governance. Each version of the document issued since 1997 has endorsed principles of shareholder primacy – that corporations exist principally to serve shareholders. With today's announcement, the new Statement supersedes previous statements and outlines a modern standard for corporate responsibility.

So this is where the whole "shareholder value" thing comes from. You might have heard about it in the past; that corporations should care first and foremost, all other things being equal, about "shareholder value."

The World Is Literally Burning

https://www.newyorker.com/cartoon/a16995

As I am talking in front of you, thousands of square kilometers in the Amazonas are being destroyed to, precisely, bring value to the shareholders of large producers and exporters of meat, soy, gold, copper, and other raw materials abundant in those regions.

At the same time injecting hundreds of millions of tons of CO2 in the atmosphere, and killing the trees that are supposed to absorb it at

the same time.

Creating a lot of value for shareholders.

Leading the world in a path that might take this planet to look more and more like Venus in the near future.

What can we do? As it turns out, there is a lot we can do.

Software Is Complex

Let me go back to what we do. We, software professionals. We know that software is essentially complex. Our business, which is to create software systems for private and public customers, is getting more and more complex every year. Sometimes even every day.

We have tighter deadlines. Higher expectations from our customers. A technology landscape that is constantly moving forward. Ideas which were good ideas five years ago quickly become legacy ideas. Lots of "beta" software to try to integrate. Bug reports. Releases. Feedback loops. Standup meetings. Agile this and agile that.

And then there is the technology itself, which is constantly shaking the floor under our feet. We just cannot stay put. We have to learn every 6 months about something new. We have to sign up for tutorials, watch webinars, attend conferences, sign up for newsletters or subscribe to RSS feeds, all in the name of staying relevant in the market.

As far as I am concerned, in my 22 years of professional software development experience, I went from Netscape to J2EE to Ruby on Rails to Node.js to Github to Stack Overflow to Mobile apps to Docker containers to Kubernetes to… whatever comes next.

There is a lot of noise out there. Can you hear the buzzing of your Twitter feed? Do you feel the guilt of not being able to follow all the things you are interested in? Do you hear that? That noise is actually blinding us from the real issues of our world.

And blinded by that noise, we have created the wrong type of software.

https://twitter.com/AriDavidPaul/status/984999126571118593

We haven't paid attention.

Some Good Changes

There is, thankfully, a great deal of good things that have happened in technology in the past 22 years.

We have an incredible choice of programming languages, IDEs, tools, debuggers, network management tools. Of course not all of them work at the same level of quality, but in average I can say that they

are much, much, incredibly much better than they used to be 22 years ago; and they will get better in the future for sure.

In spite of all the security risks online, cheaper SSL certificates means that we can ensure a decent level of privacy and confidentiality in our communications, and in turn, platforms like iOS make HTTPS communications mandatory. People are slowly adopting password managers, of which there are many and with very affordable prices. Many new devices, like high-end Android phones, Mac computers and iOS devices, are encrypted by default, making them more secure and reliable in the face of nosy governments and online threats.

Tools like Kubernetes make the cloud portable; you can run your cloud-first applications (written following the 12 factors) in your Linux, Mac or Windows laptop, on AWS, on Google Cloud, on Azure, and have those microservices up and running as fast as possible.

There is a plethora of open source code that is ready to be used. Did I say plethora? It is an amazing amount of software that is available out there. Actually there is too much software. So much that it is becoming increasingly difficult to be able to decide what to use, when and where.

And even more incredibly, we have access to all of that information wherever and whenever we want. We have small computers in our wrists and pockets with internet access. Think about this last phrase.

There is Wikipedia, Github, and Stack Overflow, which did not exist 22 years ago; these days you can tap on the collective wisdom of tens of millions of software developers in the world, all connected and sharing information about how to develop software.

What have we done with those tools?

Some Bad News

Paraphrasing a famous phrase, I think we have not done the right thing, even if we might have done the thing right.

Climate change, to begin with. I have already talked about that.

Then there are human rights violations. Some countries are openly making MITM attacks on their citizens. Was not private correspondence a right in modern democracy?

Then the state of gender inequality across the world. Women earning in average 60 to 70% of what men earn for the same work. And let us not talk about the “me too” movement, and the realization that the lives of most women are a nightmare, particularly in our industry.

The raise of far-right governments all over the world, even in so-called “democracies,” only catering for the needs of a select few, usually young, rich white men.

The negation of science and the rise of obscurantism, shown by the raise in popularity of absolutely inane ideas, such as the “flat Earth,” religious absolutism considered canon, or even worse, the reaction against the use of vaccines on kids.

Companies like Uber, AirBnB, Just Eat and many others are “disrupting” markets, leaving people homeless, jobless, eroding social conquests like paid holidays or healthcare, impoverishing our communities?

What is the world coming to?

All of the problems enumerated above are being carried like gunpowder through the platforms that we create. We, software engineers and software businessmen and software consultants and architects, we are all helping those ideas to spread.

We have a duty. We have a moral duty towards Mankind to make an ethical choice, and to take better decisions. We must build better systems, and sometimes, we have to choose not to build some systems.

The Wrong Solution

The solution is not to become more digital for the sake of it. That is the wrong solution. We need, if anything, to stop supporting ideas that do not take into account the needs of the whole world.

And very often, we need less digitalization than we think. Or, at least, we need to think if the digital solutions we come up with are the right ones.

Are we building the right software? Are we? I am seriously asking this question.

We need to start caring about each other. I say “start” and I mean it; I do not think we have written software to support and grow the right side of the equation. We have, if anything, made corporations and rich men even richer. And in return, those corporations are actually destroying the planet.

We need more than to stop caring about “shareholder value;” we need to take care of this planet, because it is the only one we have.

The Greatest Unsolved Problem

So all of this brings me to the core of my talk.

Some of us in this room are software engineers. Some others are architects. There are also consultants. Testing engineers. Quality engineers. Project managers. Business analysts. Security specialists. Many of us in this room are “agile” champions (whatever that means). We recommend tools and solutions to our customers. We

setup, develop and administer systems running in small IoT devices, smartphones, or the largest cloud providers in the world.

We use our skills to provide the best possible solution at the best possible price, in the best possible deadlines. Not an easy task, but one that we repeat day after day after day. We learn continuosly, we ask ourselves whether we are good at what we are. We suffer “impostor syndrome” every day.

And through our work, software has, indeed, eaten the world. Every company is now a software company.

We are the ones who have to start solving the greatest issues of our time. We are the ones who have the power and the intelligence and the knowledge required to solve the great problems.

Because we are the ones who know about software. We are the ones who are building the Digital Era as we speak.

And because we are the ones who know, we are the ones who can. And because we can, we must pay attention.

The greatest unsolved problem in the Digital Era, and the answer to the question in the title of this talk, is ethics. As an industry, we are simply not doing what we should be doing. And as a result, the state of the world degrades every minute.

Compared to other industries, we earn very good salaries in general. What are we doing with that money? Traveling more by plane? Buying a new car? Buying a new computer? I say we must install more solar panels. I say we must travel more by train. I say we must stop going on holidays on the other side of the world. Stop eating meat. Let us vote more responsibly with our pockets.

We have to take better decisions. We have to teach our customers to take ethical decisions. We have to be champions of ethics. The greatest unsolved problem in the Digital Era is that we are not providing value to the right shareholders.

We have to take into account accessibility in our apps. Did you know that 25% of mankind (that is almost 2 billion people) have some kind of severe visual impairment such as color blindness? Are we designing systems for them?

What about inclusion? Is the language in your apps and your web forms inclusive? Do you take into account non-binary, transgender, and LGBTQ groups into account?

What about the “Uberization” of “disrupted” industries? What about the workers in those industries? Do we realize the mess AirBnB, Uber, Just Eat and other companies are creating in the social tissue of our society? The number of families thrown into despair, poverty, even homelessness, so that a few companies in Wall Street might raise a billion or two in a flawed IPO?

And what about power consumption? Do you optimize your mobile applications to get the best the most “floating point operations per watt”? Business requirements in the age of climate change MUST include these metrics; you must test your applications in dedicated devices, to see how much batteries “suffer” when your applications are running. For example Apple provides excellent Instruments probes that allow you to verify this information during testing. Use them! Do the same evaluation of your Docker containers.

We happily decide to write yet another Electron app, although a native app would consume less energy and memory, and hence, would release less heat into the atmosphere. But no, shareholder value and all that always come to the forefront of the discussion: it is “cheaper” to create a crappy app that will burn your lap, instead of a set of efficient, targeted, coherent native apps that are much more efficient.

Writing efficient code these days is not only about having O(n log n) algorithms in the best cases; it is about not injecting more heat in the atmosphere than needed. That is the key issue.

An Oath For IT Professionals

In an article I wrote in De Programmatica Ipsum a few months ago, I proposed an oath for IT industry professionals. The title of the article is “Primum Non Nocere.” Do you know what it means? It is a Latin phrase that means “first, to do no harm.”

> Non-maleficence, which is derived from the maxim, is one of the principal precepts of bioethics that all medical students are taught in school and is a fundamental principle throughout the world. Another way to state it is that, “given an existing problem, it may be better not to do something, or even to do nothing, than to risk causing more harm than good.” It reminds physicians to consider the possible harm that any intervention might do.

I chose this title for my talk, because I think we need to apply the same principles to the making of the Digital World.

https://deprogrammaticaipsum.com/primum-non-nocere/

Here it is in its entirety:

> I swear to fulfill, to the best of my ability and judgment, this covenant:
>
> I will respect the hard-won scientific gains of those software developers and engineers in whose steps I walk, and gladly share such knowledge as is mine with those who are to follow.
>
> I will apply, for the benefit of society, all measures that are required, avoiding those twin traps of overengineering and releasing untested code.

> I will remember that there is art to programming as well as science, and that warmth, sympathy, and understanding may outweigh the knowledge of the developer or the skills of the sysadmin.
>
> I will not be ashamed to say "I know not," nor will I fail to call in my colleagues when the skills of another are needed for the proper resolution of a problem.
>
> I will respect the privacy of my users, for their lives are not disclosed to me that the world may know. Most especially must I tread with care in matters of life and death. If it is given me to solve a problem, all thanks. But it may also be within my power to generate another one; this awesome responsibility must be faced with great humbleness and awareness of my own frailty. Above all, I must not play at God.
>
> I will remember that I do not merely create a system or implement an algorithm, but I create systems for the highest benefit of society, who will have to use it and who will store their most confidential information within. My responsibility includes these related problems, if I am to solve adequately the problem at hand.
>
> I will prevent early code optimization whenever I can, for readable code is preferable to fast code.
>
> I will remember that I remain a member of society, with special obligations to all my fellow human beings, those experts in the field as well as those not initiated or knowledgeable in the matters of code and software.
>
> If I do not violate this oath, may I enjoy science and art, respected while I live and remembered with affection thereafter. May I always act so as to preserve the finest traditions of my calling and may I long experience the joy of solving the most intricate problems I am faced with.

This oath is directly inspired by the current Hippocratic Oath used by medical doctors in the United States.

Conclusion

We live in dangerous times. The time to take ethical decisions about our future is now, because the next generation will hold us responsible for them. Actually, this next generation is already here; many of you I am sure, have kids and are rightfully worried about their prospects in the future.

Sadly, "thinking digital in a digital world" is more usually associated with form than content. Ethical behaviour and ethical thinking was already a common idea in the times of Plato, and they had an absolutely

analog society back then.

Paraphrasing the Agile Manifesto, while there is value in form, we must value content more.

To put it in typical Scrum terms, we need to think beyond the Product Owner.

We need to think holistically. We need to take into account the needs of all those who are going to use our software, but we also need to take into account the needs of all those who will suffer the direct and secondary effects of our actions.

And in particular, we need to take care of the environment, because this planet is, as far as I know, the only one we have. Digital or not, there is no Planet B.

I do believe we can use our knowledge of the Digital world to start creating ethical, responsible, sustainable solutions for the next decade. I do believe in you, smart people with smart ideas and enthusiasm, to have used this fantastic day to come up with solutions and ideas that will not only bring you joy and will help you learn new things, and yes, also create lots of value for your company.

But most importantly, I believe you will be able to think ethically and to create solutions that benefit the whole of Mankind.

Remember that we are all shareholders in this planet. We need to create shareholder value for all Mankind.

Thank you very much for your attention.

About Remote Conferences

Adrian Kosmaczewski

2020-05-17

As the pandemic starts challenging all aspects of our life, I tweeted my personal opinion about software conferences in a thread.

So we've been watching bits and pieces of @theRemoteCon with my colleagues today, and I have a few things to say about the format; not about the content, and this might be useful for other upcoming online conferences (like @appbuilders_ch :)

— Adrian Kosmaczewski □ □□ (@akosma) April 7, 2020

Here's the thread, unrolled and summarized.

1. Short talks. No more than 10 minutes.
2. Pre-recorded. Speakers record themselves giving the talk. The recording should be YouTube-quality; that is, no "uhmmmmm" or "ehhmmmm". Also: no music.
3. The speaker's face appears on the recording, on a small window above the slides. Let us see you. At all times. That's important. Gestures, smiles.
4. The most important part of this idea: the speaker only presses "Play" and switches immediately to the associated chat system, answering questions from the audience, while the talk happens. That's right! Q&A session right during the session. Now that's value imho.

The advantages of this approach are:

1. Vetoing of talks beforehand. A small step for the organizers when talks are only 10 min long, a huge step for the attendees.
2. Optimized delivery, particularly when demoing, which might be useful in tech conferences.
3. Reuse. Yes, a speaker might be able to reuse a talk later in another context, why not.
4. Deliverables. Forget the PDF-only deliverable; there's also a video with all the talking.
5. Higher density of information. Remove those "uhhmmmm" and "eehhhmms" and silences.
6. Respect of the timetable. There will be never again a speaker stepping on the timeslot of the next one.
7. And for accessibility, since the talks are pre-recorded, don't forget the subtitles. This is super important.

Reboot

Adrian Kosmaczewski

2020-11-01

I started my first blog in December 2004. But that was not my first website. I published that one in August 1996. At first, just a pure HTML website, with almost no JavaScript or CSS, lots of <FONT> and <FRAME> and <MARQUEE> tags all over the place.

I actually used an extension of Microsoft Word 6.0 to create those first webpages. Hard to believe.

I want to write, hence it is time to start over. I build that first blog using Movable Type[1], and then I migrated it to WordPress[2] around 2006. I have been a faithful WordPress user since then. But I am tired of it. Too much infrastructure for just a few words.

So for this new one, I choose a simpler option. Pure text. Just like my first website, this one will be pure HTML, and of all options, I chose Hugo[3] as my platform of choice.

There are several things that I like about Hugo. To begin with, it is written in Go, it is thus portable and extremely fast. I like command-line tools written in Go.

Also, there are lots of nice themes available for Hugo (I am no designer, so I definitely need help with that.)

And last but not least, Hugo supports Markdown (of course) and Asciidoc. Since writing my first book I became a big fan of Asciidoc, and lately, of Asciidoctor[4].

I can write my prose in any text editor I like (this text was typed on Emacs, for example) and then I can deploy the result using `scp`. The simplest approach I could find.

There is a certain peace of mind when using simple tools that work well together. What developers usually call “the Unix way”, I guess.

I also like the “live preview” functionality of Hugo. Just write, save, and boom, the browser refreshes itself automatically. This is terrific.

[1] https://movabletype.org/
[2] https://wordpress.org/
[3] https://gohugo.io/
[4] https://asciidoctor.org/

I will be writing a new article here every week. There will be some technology, some personal thoughts, some stories. But no comments.

Thanks for reading.

Migrating from macOS to Linux

Adrian Kosmaczewski

2020-11-07

This is the story of how, after being a loyal macOS user for 15 years, I decided to start using Linux full-time.

This article provides a detailed report of all the small decisions I took from late 2016 until late 2018, the moment in which I got my TUXEDO Computers laptop with Ubuntu 18.04 pre-installed.

The Situation

For context, I will describe my entaglement with the Apple platform around 2016. Entanglement is the word, yes. The level of intrusion that tech companies can have on one's life and production is staggering.

I started using Mac OS X Jaguar in December 2002, when I bought my first iBook. I still have that white machine, and it still boots like the first day, even if the battery is utterly dead. Before that I was a Windows XP user, having used (and programmed for) all versions of Windows since 3.1.

The reason I bought that iBook was because I wanted to learn 2 things:

- Objective-C, and
- The Unix command line

That 2002 iBook was not my first Mac. Such honor belongs to a PowerBook 150 I bought in 1994 while studying in the University of Geneva. That little machine (Motorola 68030 CPU, 2 MB RAM, 40 MB hard disk) was a good companion during my studies. But it was not by far my main machine. In any case I did not write any code in it, even though a friend installed a copy of Metrowerks CodeWarrior[1] in it. Sadly, at some point in 1997 it stopped working, and then I got rid of it.

In 2008 I started a career as a full-time iOS and Objective-C developer. Those were the glorious times of Mac OS X Snow Leopard. It was the peak of stability, usability and fun for Mac OS X; we might never see anything like that again. Maximum productivity and zero downtime.

As a developer, Xcode 3 was a joy to use. Objective-C 2.0 was all the rage, with `@property` and `@synthesize` and the whole package. It

[1] https://en.wikipedia.org/wiki/CodeWarrior

compiled in milliseconds, executed fast, and one could link it to any C or C++ library out there.

Of course, it went all downhill from there. It started with the train-wreck that was Xcode 4. Sensible souls will remember.

Following that trend, by 2016 the quality of the whole Apple offering was so terrible that I decided to start a migration towards something else. The tipping point was the purchase of the crappiest, most expensive laptop I ever wasted money on: a 13" late 2016 MacBook Pro with a useless TouchBar and the worst keyboard I have ever had the bad luck to type in a laptop. Enough was enough.

So I decided to leave the Apple galaxy. The question was, to go where? How? Leaving macOS (as Mac OS X got renamed in the meantime) meant a huge transfer of data from one galaxy to some other.

Platform lock-down is an actual thing, and it is costly.

And I found out I was not the only one[2] in this path.

After evaluating the standard trinity of options outside Mac OS X, namely Windows, Linux and BSD (NetBSD[3] in particular), I decided that I wanted to use Linux full time. In particular, I settled for Ubuntu. I had already been playing with it in virtual machines and Raspberry Pi boards since 2006, and I knew the system quite well. I felt comfortable enough with it.

(I actually considered Oracle Solaris[4] as well, but for a short while. Like, for thirty seconds.)

One does not simply jump ship just like that. There are three basic variables to take care of in such a migration: Hardware, Software, and Data. In the next sections I will provide details about each.

A Note about Ideology

I want to be very clear: I have not chosen Linux out of a sense of doing "the right thing" or "making the world a better place" or "belonging to a community" or anything like that. I do not think that open source or free software are better or worse than commercial software. I will (maybe) explain this position in a future post.

What I do care more about, if anything, is the format in which I store the documents I create during my computing. If there are formats whose specification is not owned by a corporation, and whose standards are publicly available, I choose them. Otherwise, I go with commercial alternatives, because sometimes you just got to get stuff done.

[2] https://bitcannon.net/post/a-year-away-from-mac-os/
[3] https://netbsd.org/
[4] https://www.oracle.com/solaris/

Of course, I care for a certain quality of the software I use. I experience the same proportion of crappy software in FOSS and COTS. The world is not a better place because of the existence of any of these.

All of this means that my choices of operating system are related to my computing needs and knowledge at a particular time in my life. Windows was OK to begin with in the 1990s; Macs were OK during a certain period of time; and now Linux is also good enough for me. Maybe in a few years I will be using NetBSD for the same reasons.

1. Hardware

This is the part of the trip that took me the longest. I wanted to find a good machine that ran Linux without issues, so I started asking around in Twitter to gather some opinions about the subject.

I really, really, really did not want to bother with recompiling kernels, device drivers, or editing `/etc/whatever/conf.rc` following the contradicting advice of a bunch of Stack Overflow answers. Seriously. Thanks, but no thanks.

I wanted to find a good machine with good Linux support. I knew this was possible. I just needed to do some research. And I did find quite a few suitable candidates among the big brands:

- Dell XPS 13''[5], with 16 GB RAM, with raving reviews online, and sold in Switzerland with US keyboard and Ubuntu pre-installed. (US keyboards are a mandatory requirement for me.)
- Lenovo X1 Carbon[6], widely praised online as Linux compatible. It actually bundles a dedicated "Linux" compatibility mode in the BIOS settings, and receives official upgrades from Lenovo. I learnt that it was compatible with Arch Linux[7] and had also got official Ubuntu Certification[8]. And then one day I held one of these in my hands, and are they lightweight! What a lovely construction. And what a great keyboard. I was amazed.

Among my friends online, Fernando Cejas[9] recommended a Lenovo Thinkpad T460, similar to this T470[10] with support for 32 GB of RAM and two hard disks. Fabrice Truillot[11] started using a ThinkPad T480s around 2018, also marked as compatible with Linux.

Beyond the big PC makers, I discovered a myriad of independent computer makers specialized in Linux-compatible hardware:

[5] http://www.dell.com/ch/p/xps-laptops?c=ch&cs=chbsdt1&l=fr&s=bsd&~ck=mn

[6] http://shop.lenovo.com/ch/de/laptops/thinkpad/x-series/x1-carbon/

[7] https://wiki.archlinux.org/index.php/Lenovo_ThinkPad_X1_Carbon

[8] https://certification.ubuntu.com/hardware/201702-25372/

[9] https://twitter.com/fernando_cejas/status/940218989711634432

[10] https://www3.lenovo.com/ch/de/laptops/thinkpad/t-series/ThinkPad-T470p/p/22TP2TT470P?menu-id=T470p

[11] https://twitter.com/fabricetdc/status/965921961879785472

- System76[12], Purism[13], and EmperorLinux[14] in the USA.
- Slimbook[15] in Spain.
- And finally TUXEDO Computers[16] from southern Germany (recommended to me by Aral Balkan[17] himself!)

The nice things about these brands is that they are independent computer makers; supporting small businesses is important to me.

And of course, there were those smaller and extremely cheap Linux-based laptops based on the ARM architecture came to the list:

- Pinebook Pro[18]
- Pi-Top[19]

Interesting but a bit underpowered for my taste. Anyway, I kept all of this information in a Joplin note, and evaluated all the options.

Peripherals

Of course I had a bunch of peripherals around my Mac, and I wanted to make sure they were compatible.

For my Logitech wireless mouse, Solaar[20] and Piper[21] provide some functions similar to that of the Logitech Options[22] software.

What about my Logitech R700 wireless presenter[23]? No problem. Logitech C930e webcam[24]? Check. Logitech MX Master 3 mouse[25]? Ready. (Yes, I am a happy Logitech customer.)

Finally I discovered that HP had printer software for Linux[26] compatible with my HP Color LaserJet Pro MFP M277dw printer.

So far, all looked good.

[12] https://system76.com
[13] https://puri.sm/
[14] http://emperorlinux.com/
[15] https://slimbook.es/
[16] https://www.tuxedocomputers.com/
[17] https://twitter.com/aral/status/1006222602250092544
[18] https://pine64.com/
[19] https://www.pi-top.com/
[20] https://github.com/pwr-Solaar/Solaar
[21] https://github.com/libratbag/piper/
[22] https://www.logitech.com/en-us/product/options
[23] https://www.logitech.com/en-roeu/product/professional-presenter-r700
[24] https://www.logitech.com/en-us/product/c930e-webcam
[25] https://www.logitech.com/en-us/products/mice/mx-master-3.910-005620.html
[26] http://hplipopensource.com/

Final Choice

I decided to get a TUXEDO InfinityBook Pro 14 v5[27] laptop with a 64-bit Intel i7 CPU, 32 GB of RAM, and two SSDs. It came bundled with Ubuntu 18.04.

The TUXEDO people are located not far from the Swiss border, and sell their computers with Swiss chargers and deduct the German VAT automatically from the purchase price. It could not be better. The whole buying experience with them was flawless.

They provide full support, and this even includes a USB stick with the Windows drivers for your machine, in case you might want to have a dual-boot setup. I do not have such configuration, but I think it is a thoughtful touch.

After two years, I can get 5 hours in average of full-time use on a full charge of the battery with lots of apps running. The keyboard of this machine is fantastic. I recently upgraded this machine to Ubuntu 20.04, without absolutely no issue at all.

In the connectivity side, it has a built-in Ethernet port, two USB-A ports, one USB-C port (which can be used for charging the battery, and to connect to their docking station[28]), and a built-in SD card reader. The trackpad is good enough (I seriously do not care that much about it) and everything just works: webcam, built-in mic, etc.

To be honest, the weakest point of the machine are the built-in speakers. They are really quite crappy, but the truth is, I simply do not use them. If you fancy listening to music out loud in your Mac, then you might not like this machine. I personally do not really care. I have my old USB Harman Kardon SoundSticks bought in 2003 and they work like a charm.

In terms of stability and speed, in many ways it feels a lot like using a 2009 Mac with Snow Leopard once again. It is super, super, super stable.

To top it off, I started working at VSHN[29] a few months later, and they got me a fantastic Lenovo ThinkPad X1 for work. So now I have my two preferred Ubuntu machines, one for work, another for personal use.

2. Software

As I mentioned above, I chose Ubuntu as the operating system for this new phase. The main reason is the large (very large) quantity of

[27] https://www.tuxedocomputers.com/en/Linux-Hardware/Linux-Notebooks/10-14-inch/TUXEDO-InfinityBook-Pro-14-v5.tuxedo

[28] https://www.tuxedocomputers.com/en/Linux-Hardware/Accessories-books-co-/USB-accessories/Universal-docking-station-for-all-TUXEDO-Books-Type-C-Type-A-USB-connection_1.tuxedo

[29] https://vshn.ch/en/

software available for it. Pretty much every single open source project out there offers a `.deb` package ready to be installed. Ubuntu is really popular. That, in itself, is a big bonus.

Now I needed to find suitable software equivalents for all the things I did on my Mac.

Installing Software

There are a myriad of ways to install software in a Linux machine, beyond the usual `sudo apt install` or `sudo make install`. There is even `umake`[30] which you might not have heard of before. Furthermore, these days most popular programming languages come with their own package managers: PIP[31], Conan[32], Composer[33], npm[34], go get[35], Cargo[36], Maven[37], dotnet add[38], Rocks[39], Swift Package Manager[40]... as if there were not enough of them.

You could also download a `.deb` package and simply `dpkg -i` it, or double-click on it. Getting used to all of these ways to install software was very important for me.

Actually, there is even Homebrew[41] for Linux now. I tried it for a while but I do not use it anymore.

But then I discovered something new: ready-to-use applications; just download and use them:

- AppImage[42] has become kind of a standard lately. Very popular with FOSS desktop apps; just download the `.appimage` file, then `chmod +x` it, and run it.
- Flatpak[43] is slowly getting traction as an alternative to AppImage.
- Finally, Snaps[44] are an Ubuntu-only solution for the same problem. Very useful and lots of excellent apps are ready to be installed with it.

[30] https://github.com/ubuntu/ubuntu-make
[31] https://pypi.org/project/pip/
[32] https://conan.io/
[33] https://getcomposer.org/
[34] https://www.npmjs.com/
[35] https://golang.org/pkg/cmd/go/internal/get/
[36] https://doc.rust-lang.org/cargo/
[37] https://maven.apache.org/
[38] https://docs.microsoft.com/en-us/dotnet/core/tools/dotnet-add-package
[39] https://luarocks.org/
[40] https://swift.org/package-manager/
[41] https://brew.sh/
[42] https://appimage.org/
[43] http://flatpak.org/
[44] https://snapcraft.io/

Note-Taking Apps

Note-taking is a basic component of my computing needs. I usually keep my notes application open at all times, searching and writing notes every time I figure out how to do anything.

I need my note-taking application to sync notes across desktop and mobile platforms, because I like to quickly create notes with audio or images when I am on the go.

And when I say mobile, I do mean both iOS and Android; I used both in the past, and I plan to use them in the foreseeable future. There are things I like about iOS, and then there are things I like about Android.

I had been a paying Evernote user from 2012 to 2014, at which moment I moved to Apple Notes, which had a decent set of functionality for what I needed. It also synchronized stuff reasonably well across iOS and Mac devices. Evernote was bloated, slow, and crashed too much for my taste. Apple Notes crashed, too, but less often, and was simpler and faster.

But Apple Notes had a major drawback: there was no "export" function whatsoever.

So first I had to find the location of notes in Mac OS X (hint: they were stored in `~/Library/Group Containers/group.com.apple.notes/`) and tried to figure out ways to bulk-export all of my data.

It was such a mess, I ended up exporting only the most important notes from my collection. It was enough, and convinced me to never again deal with Apple Notes.

For a short while in 2016 I used Bear[45], a fantastic Mac application that had the two most features I looked for: Markdown support and PDF export. It did look good, too, but seriously by that time I cared more about functionality than looks. I wanted software that worked.

In my search for the perfect note application I considered these options:

- Simplenote[46], a bit too simple for my taste.
- Microsoft OneNote[47], another bloated mess like Evernote.
- Tusk[48], and Electron-based third party client for Evernote.
- Boostnote[49]: interesting, but it was under heavy development when I tried it. PDF export was not yet available, the iOS apps did not yet sync over Dropbox, and the iPad app had no landscape or iPad Pro support. I think it has gotten better now, but I cannot tell.

[45] https://bear.app/
[46] https://simplenote.com/
[47] https://www.onenote.com/
[48] https://github.com/klaussinani/tusk
[49] https://boostnote.io/

- nb[50]: Full CLI, written in Bash, uses Git for sync, support for all Pandoc formats, support for bookmarks and images and docs, one git repo per notebook.
- And finally one day I discovered Joplin[51] by pure chance.

In January 2017 I chose Joplin. It is cross-platform (including a great terminal app), has several sync mechanisms (including Dropbox), and has really good mobile apps. I have been using it every single day for over three years, and I cannot state how happy I am, and how good it is. I use it for everything, all the time.

Here is how I moved all of my notes from Bear into Joplin:

1. Exported notes from Bear in backup format.
2. Opened the resulting `bearbk` file as a zip file and unzipped it.
3. Ran the following script, and at the end of the process, the contents of the text of all notes were stored in a folder named `_export`:

```
#!/usr/bin/env ruby

export = "_export"

Dir.mkdir(export)

Dir.foreach('.') do |item|
  next if item == '.' or item == '..' or File.file?(item) or item == export
  filename = export + '/' + item.sub('.textbundle', '.md')
  contents = File.read(item + '/text.txt')
  File.open(filename, "w") do |f|
    f.write(contents)
  end
end
```

At this point in time, I consider Joplin to be simply perfect for my needs. The only thing that I found flaky in it was the whole encryption thing. All of a sudden it restarts reencrypting all of your notes and god knows why. Other than that, it is great.

Web Browser

Having been a loyal Safari user since version 1.0, I started using Firefox as my main browser mid-2016. At that time I was using not only iOS devices but also Android devices, and I wanted to use a web browser that had some synchronization feature shared among all of these platforms.

I have used and liked Firefox since 2004. I still do. And of course, thanks to it I could migrate my browsing history and preferred extensions from the Mac to my Linux box without any issues.

[50] https://xwmx.github.io/nb/
[51] https://joplinapp.org/

By the way, I have an iOS device, and since I upgraded to iOS 14, my main browser is Firefox for iOS. Finally we can set a default browser in iOS. It took Apple 12 years to add that feature.

I have also Chrome[52] and Opera[53] installed in my Linux machine, but only for testing websites and web apps. And I will install Edge[54] when it comes out of beta, too. Although all of these use the same rendering engine these days, so it does not really make sense to have them all.

Chrome is the new Internet Explorer.

Programming Languages

The great thing about Linux is that virtually every open source programming language is available for it. I have (of course!) installed the GNUstep[55] toolchain to have access to Objective-C, a language that I like a lot.

But not only that: in the past two years I used, learnt or played with Go[56], GNU Smalltalk[57], Common Lisp[58], FreeBASIC[59], Free Pascal[60], Rust[61], GnuCOBOL[62], Rexx[63], and so many more, and so many others will come.

And yes, there is Swift for Linux[64], if you are into that kind of thing. Not for me though.

Music

This one was rather easy. I started using Spotify in 2013; then switched to Apple Music for a few months in 2015. But it was really bad. It was “click play and wait for hours for the music to start” level of bad, and did not even have all the music I wanted to hear! So I quickly went back to Spotify; and the good thing is that Spotify has a great web app and a native client for Linux. Boom.

Spotify is so good, it is hard to describe. I even have Spotify in my LG TV now. It is everywhere.

---

[52] https://www.google.com/chrome/index.html
[53] https://www.opera.com/
[54] https://blogs.windows.com/msedgedev/2020/10/20/microsoft-edge-dev-linux/
[55] http://gnustep.org/
[56] https://golang.org/
[57] https://www.gnu.org/software/smalltalk/
[58] https://common-lisp.net/
[59] https://freebasic.net/
[60] https://www.freepascal.org/
[61] https://www.rust-lang.org/
[62] https://gnucobol.sourceforge.io/
[63] https://regina-rexx.sourceforge.io
[64] https://swift.org/

I also have a rather large MP3 collection, and since I was fed up with iTunes, I started using Musikcube[65] instead. It is a lovely player and it is amazingly good. I started using it on the Mac, and then continued using on Linux. Simple and good.

Text Editors and IDEs

One of the most valuable things I learnt using Mac OS X was, ironically, the Unix command line; and with that came Vim[66] and GNU Emacs[67]. After the TextMate 2.0 fiasco I started using MacVim a lot (with Janus[68], then I moved to terminal `vim`, and by 2017 I became a diehard Microsoft Visual Studio Code user.

The good thing of `vim`, `emacs` and Visual Studio Code is that they work perfectly well in Linux. I do not use joe[69] or nano[70] much.

As for serious software development, I needed some IDEs, so I got myself a full JetBrains subscription[71], and apart from AppCode (understandably only available in macOS) I have access to fantastic software for my work. Their software is among the best and most stable ones I have ever used.

Video

Even when using Mac OS X, I always found VLC[72] much better for video consumption than QuickTime; VLC simply plays anything I throw at it. QuickTime would always complain about some missing codec or whatever.

In the command line world of Linux I discovered `youtube-dl` to download pretty much anything I found online, `ffmpeg` to convert between formats, and `mplayer` to play them back. All works perfectly well. I could even open old videos I had kept from the 1990s, stuff I had not seen in decades.

Funny tip: `mplayer -vo caca -quiet -noborder MovieName.avi` to play a video directly on your terminal. It requires `mplayer` built `--with-libcaca --with-libdvdnav --with-libdvdread` and the of course the end result is unwatchable. Who cares!

Anyway, I digress. These days I am starting to edit my own videos, and for that I found OBS Studio[73], which I did not know, plus a lot more software I tried to edit videos:

[65] https://musikcube.com/
[66] https://www.vim.org/
[67] https://www.gnu.org/software/emacs/
[68] https://github.com/carlhuda/janus
[69] https://joe-editor.sourceforge.io/
[70] https://www.nano-editor.org/
[71] https://www.jetbrains.com/store/
[72] https://www.videolan.org/vlc/
[73] https://obsproject.com/

- Shotcut[74], based on Qt. I used it for the first VSHN.timer video, and promised myself to never use it again. It crashed all the time.
- OpenShot[75] is much more stable than Shotcut, and the story[76] of the developer behind it is brilliant: C#/.NET developer without experience in Linux, started the project in May 2008… and here it is now. The project includes C++ libraries for video and audio processing. I edited the second VSHN.timer video with it, and it was OK-ish, so I kept looking for other options.
- Flowblade[77] is much more solid than the previous two options, and I will use it for my future VSHN.timer videos.
- DaVinci Resolve[78] is a Hollywood-level thing, apparently, but lots of Ubuntu users have reported issues[79] during installation, and apparently it only works well in CentOS. I have found more installation problems[80] if you are curious. In my case I tried to install it, and I could, but it did not start at all, complaining about the ThinkPad GPU. Whatever.
- Other options I found are Lightworks[81] and KDEnlive[82] but I have not tried them yet.

Update, 2021-01-20: Add Pitivi[83] to the list above. Also, I'm not using Flowblade anymore for VSHN.timer, having found KDEnlive to be a much better alternative. By better I understand: stable, fast, and compatible with Windows.

Update, 2021-07-30: More about video in Linux[84] in this blog.

Video editing is certainly an area where the Mac has a large advantage. If you are a video professional, I would certainly not recommend you to move to Linux.

Image Editors

I am not a visual designer, so my needs for graphics are very limited; just a bit of retouching here and there, and converting images into other formats. In those cases Gimp[85] and Inkscape[86] work perfectly well for me, and they support all of the formats I could dream of using.

[74] https://www.shotcut.org/
[75] http://www.openshot.org/
[76] https://www.openshot.org/story/
[77] http://jliljebl.github.io/flowblade/
[78] https://www.blackmagicdesign.com/products/davinciresolve/
[79] https://forum.blackmagicdesign.com/viewtopic.php?f=21&t=58668&p=337953&hilit=ubuntu+Andr%C3%A9Rodrigues#p336664
[80] https://forum.blackmagicdesign.com/viewtopic.php?f=21&t=56878
[81] https://www.lwks.com/
[82] https://kdenlive.org/
[83] http://www.pitivi.org/
[84] /blog/video-editing-in-linux/
[85] https://www.gimp.org/
[86] https://inkscape.org/

Krita[87] is another excellent option to Gimp. I am using all of them, depending on what I have to do.

I have configured Gimp to use Photoshop menus and shortcuts[88] which helped a lot at first. But then I ended up learning the actual native shortcuts. Muscle memory takes a little while.

As for the few Pixelmator[89] files I had, I converted them to PSD format (Photoshop), and I could open them in my Linux laptop without problem.

Password Manager

When I started my migration progress there was no 1Password Linux client, so I moved to Enpass[90] instead. They have great mobile and desktop clients for all operating systems, and good browser plugins as well. And it synchronizes over Dropbox.

Update, 2020-11-21: Following a recommendation[91] by Tobias Brunner[92] I have switched to KeePassXC[93], synchronizing my data with Strongbox[94] over Dropbox.

Communications

Thankfully I had stopped using iMessage with friends and family around 2012; since that time we were all communicating in Telegram. So that was another easy switch. Telegram has both web-based and Linux clients, so I can be notified of messages in all of my devices without issues.

There are Skype and Zoom clients for Linux; no need to change anything there.

Other Apps

To make a long story short, this is the full list of “equivalences” I found for many apps I used in Mac OS X:

Apple applications:

- Notes: Joplin[95]
- Safari: Firefox[96]

[87] https://krita.org/
[88] https://www.linuxuprising.com/2018/11/configure-gimp-210-to-use-photoshop.html
[89] https://www.pixelmator.com/pro/
[90] https://www.enpass.io/
[91] https://tobru.ch/newsletter-26/
[92] https://tobru.ch/
[93] https://keepassxc.org/
[94] https://strongboxsafe.com/
[95] https://joplinapp.org/
[96] https://www.mozilla.org/en-US/firefox/new/

- Mail: Thunderbird[97] or Evolution[98]
- Music: Spotify for Linux[99]
- iTunes: Musikcube[100]
- iMessage: Telegram Desktop for Linux[101]
- iWork Suite: LibreOffice[102] and there is even AbiWord[103] in case you have old formats to open, and WordGrinder[104] for a full command-line experience.
- Tasks: Microsoft To-Do[105]
- iBooks: Calibre for Linux[106]
- AirDrop: Resilio Sync[107]
- QuickTime: VLC[108] for playback, and SimpleScreenRecorder[109] for screen recording
- GarageBand: Audacity[110]
- iMovie: Flowblade[111]
- iPhoto: Darktable[112]
- iCloud Drive: Dropbox[113]

Third-party software:

- 1Password[114]: Enpass[115]
- Dash[116]: Zeal[117]
- IDEs: JetBrains[118]
- TeXShop[119]: TeXstudio[120]
- Transmit[121]: FileZilla[122]
- VirtualBox: VirtualBox for Linux[123]

[97] https://www.thunderbird.net/en-US/
[98] https://wiki.gnome.org/Apps/Evolution
[99] https://www.spotify.com/ch-de/download/linux/
[100] https://musikcube.com/
[101] https://desktop.telegram.org/
[102] https://libreoffice.org/
[103] https://abiword.en.softonic.com/?ex=CORE-117.4
[104] http://cowlark.com/wordgrinder/index.html
[105] https://to-do.microsoft.com/
[106] https://calibre-ebook.com/download_linux
[107] https://www.resilio.com/individuals/
[108] https://www.videolan.org/vlc/index.html
[109] https://www.maartenbaert.be/simplescreenrecorder/
[110] https://www.audacityteam.org/
[111] http://jliljebl.github.io/flowblade/
[112] http://www.darktable.org/
[113] https://dropbox.com
[114] https://1password.com/
[115] https://www.enpass.io/
[116] https://kapeli.com/dash
[117] https://zealdocs.org/
[118] https://www.jetbrains.com/
[119] https://pages.uoregon.edu/koch/texshop/
[120] https://www.texstudio.org/
[121] https://www.panic.com/transmit/
[122] https://filezilla-project.org/
[123] https://www.virtualbox.org/wiki/Linux_Downloads

- Slack: Slack for Linux[124]
- Twitter: Corebird[125]
- Deckset[126]: first GitPitch[127], then Asciidoctor-Reveal.js[128]
- And cool-retro-term[129] instead of iTerm2[130] but just for the sake of fun :)

Terminal apps that stayed untouched across Mac and Linux:

- zsh[131], emacs[132], vim[133], tmux[134], irssi[135], mc[136], w3m[137], htop[138], tig[139], musikcube[140], wordgrinder[141], sc-im[142], k9s[143], asciinema[144], asciidoctor[145], pandoc[146], git[147], git-extras[148], onefetch[149], podman[150], dosemu[151], xdotool[152], caddy[153], bat[154], v4l2-ctl[155], gphoto2[156], pyenv[157], rbenv[158], nvm[159], curl[160], ag[161]…

By the way, here is a tip which might help when moving from Mac to

[124] https://slack.com/intl/en-ch/downloads/linux
[125] https://corebird.baedert.org/
[126] https://www.deckset.com/
[127] https://gitpitch.com/
[128] https://asciidoctor.org/docs/asciidoctor-revealjs/
[129] https://github.com/Swordfish90/cool-retro-term
[130] https://iterm2.com/
[131] https://www.zsh.org/
[132] https://www.gnu.org/software/emacs/
[133] https://www.vim.org/
[134] https://github.com/tmux/tmux/wiki
[135] https://irssi.org/
[136] https://midnight-commander.org/
[137] http://w3m.sourceforge.net/
[138] https://htop.dev/
[139] https://jonas.github.io/tig/
[140] https://musikcube.com/
[141] http://cowlark.com/wordgrinder/index.html
[142] https://github.com/andmarti1424/sc-im
[143] https://k9scli.io/
[144] https://asciinema.org/
[145] https://asciidoctor.org/
[146] https://pandoc.org/
[147] https://git-scm.com/
[148] https://github.com/tj/git-extras
[149] https://github.com/o2sh/onefetch
[150] https://podman.io/
[151] http://www.dosemu.org/
[152] https://github.com/jordansissel/xdotool
[153] https://caddyserver.com/
[154] https://github.com/sharkdp/bat
[155] https://www.mankier.com/1/v4l2-ctl
[156] http://gphoto.org/
[157] https://github.com/pyenv/pyenv
[158] https://github.com/rbenv/rbenv
[159] https://github.com/nvm-sh/nvm
[160] https://curl.haxx.se/
[161] https://geoff.greer.fm/ag/

Linux; install xclip[162] and copy this in your `~/.aliases` file:

```
alias pbcopy="xclip -selection clipboard"
alias pbpaste="xclip -selection clipboard -o"
```

I hope I am not missing anything in the list above. Feel free to use this as the basis for your own migration matrix.

3. Data

I was a hardcore user of most Apple services. As a paying iCloud user, I used it for all of my digital life. That was, after all, the whole idea of it. But the user experience was awful. All too often I did lose data because of the failures of iCloud to properly synchronize data between devices made by the same company. Unacceptable, and too often, and for a ridiculously high price.

I moved all of my files from iCloud to Dropbox. I evaluated many other options, but Dropbox had many positive points to it:

- I had already used it since 2008, and it always worked well and fast.
- Best synchronization ever. Seriously unbeaten, particularly after comparing it to iCloud (of course!) but also OneDrive and Box.
- They have great mobile clients; I used the Android and iOS version and I liked them both.
- And last but definitely not least, there is a Dropbox Linux client[163].

In terms of local storage, Ubuntu is able to access my external hard drives formatted with HFS+[164] without issue.

Documents

In my life as a conference speaker, I used Keynote extensively from 2006 to 2013. In 2014 I discovered Deckset[165], which uses Markdown for slides. Of course there is no Deckset in Linux, but I moved my workflow to Asciidoc + Reveal.js[166] slides.

Fortunately, Pages and Numbers documents can be easily converted into Microsoft Word and Excel files, which are completely compatible with LibreOffice, so I did that too.

Well, almost; until a few years ago, the latest version of Apple Pages for Mac or iOS could not open files created in… Apple Pages 1.0 (circa 2005). No comments. Thankfully they added support for that format again at some point. I still can not believe I just wrote this paragraph.

[162] https://github.com/astrand/xclip
[163] https://www.dropbox.com/install-linux
[164] https://en.wikipedia.org/wiki/HFS_Plus
[165] https://www.deckset.com/
[166] https://asciidoctor.org/docs/asciidoctor-revealjs/

E-Books

After suffering the ordeal of dealing with Apple iBooks, I decided to never buy books from Apple ever again. I only had a few books in "iBooks" format, and of course I will not be able to read them ever again. It is OK, that will teach me a lesson.

I had a few Kindle books, thankfully not a lot; I converted them into EPUB (using Calibre plugins) and coalesced all of my book collection into a great Calibre collection of PDFs and EPUB files.

For the few CHM format books I had (from my times as a .NET developer), xCHM[167] works fine.

I finally bought a Kobo Aura H2O[168] e-book reader in 2016, and since then I have been happily reading books in EPUB format. Calibre recognized it immediately both on the Mac and on Linux. Plug and play.

I also took the decision of not buying any more books for Amazon; I only use their website to find the book I want, and then I buy it directly from the publisher. Most publishers offers EPUB without DRM and in various formats (including PDF and EPUB, and even Kindle `.mobi` files, although I do not have a Kindle.) The last e-book I bought from Amazon was in 2012.

EPUB files without DRM are the kind of e-book I pay for.

Did Not Happen

As I mentioned previously, I did not migrate to Linux following a false sense of nerdiness, belonging to a community, or some otherwise grandiose feeling of betterment of the world. There are, then, quite a few things that many other Linux users choose to do, that I do not.

Using mutt[169] for mail for example. Or rainbowstream[170] for Twitter, or newsboat[171] for RSS. Or using Org Mode[172] in Emacs.

I sometimes do some command-line web browsing, but for that, I prefer w3m[173] to lynx[174].

Other things that did not happen:

- Replacing SourceTree[175] with GitKraken[176] or SmartGit[177].

[167] https://github.com/rzvncj/xCHM
[168] https://us.kobobooks.com/products/kobo-aura-h2o-edition-2
[169] http://www.mutt.org/
[170] https://github.com/orakaro/rainbowstream
[171] https://newsboat.org/
[172] https://orgmode.org/
[173] http://w3m.sourceforge.net/
[174] https://invisible-island.net/lynx/
[175] https://www.sourcetreeapp.com/
[176] https://www.gitkraken.com/
[177] https://www.syntevo.com/smartgit/

- Replacing Ulysses[178] with Document Node[179].
- Replacing Marked[180] with Caret[181].

Lots of Mac-only software is still unbeatably good. That is just the way it is. For the three last items, I ended up replacing them with Visual Studio Code extensions, or by JetBrains IDEs plugins.

Conclusion

My migration to Linux started in January 2017 and ended in December 2018, when I got my TUXEDO Computers laptop on the mail, and got rid of my last MacBook with TouchBar.

My faithful Logitech webcam, my USB Yeti microphone, and my HP laser printer, all worked out of the box with Linux. And even more ironic, my 2002 Harman Kardon speakers did too. I have two standard monitors that have cost me less than 200 USD each, and they work flawlessly with my two Linux laptops.

I have a productive setup, with a machine with a decent battery life, great specs, all at an unbeatable price (below 2000 USD). My files are saved in standard, open formats, and I will be able to open them in the future.

During the last two years, this setup has given me much, much less headaches than what macOS was giving me in 2016. It is far from perfect; but it is surprisingly stable, efficient, fast, and oh my god, so much cheaper.

Every user has a different mileage, so I do not advocate migrating to Linux for everyone. Being a developer certainly helped me to find quirks and solve problems here and there. But in general, I had much less issues than I thought I would have. Like, far less. I was pleasantly surprised by Linux.

I hope this guide will help you jump to Linux if you feel the curiosity to learn something new. I have learnt so many new things exploring Linux, I am very happy with it so far.

As far as I am concerned, my next step will be to migrate to NetBSD, but not in the immediate future.

PS: I am not interested in receiving opinions about how I could have taken any of the decisions above differently. Please do not contact me to let me know your point of view or to make suggestions. I do not care and the fact that there are no comments in this blog should give you a hint about the fate of any e-mail you might want to send to me.

[178] https://ulysses.app/
[179] https://documentnode.io/
[180] https://marked2app.com/
[181] https://caret.io/

VSHN.timer

Adrian Kosmaczewski

2020-11-13

Since August 2019 I took the duty of publishing a weekly series of blog posts called "VSHN.timer[1]".

The idea of VSHN.timer is to extract, from all the links shared by my colleagues in the VSHN[2] chat system, the best five in any particular subject, of course always centered around technology, DevOps, Kubernetes, and other cloud-related stuff.

Thankfully, so far, the feedback from our readers has been nothing short of terrific.

So that motivated us, earlier this year, as the pandemic started looming across the planet, to take another small step. Following the request of a few readers, we made an e-mail newsletter[3] out of it, one of those gazillion newsletters you can subscribe to. I myself am subscribed to a few of those (TLDR[4] is undoubtedly the best), and I have to say I like having someone recommending links to me. And I like short newsletters. I hope VSHN.timer is short enough. And I hope it is of enough quality.

But as if that were not enough, lately we have added yet another channel to our distribution: YouTube[5]. That is right, a new video every Monday, with the same five links, but with the added bonus of that unmistakable latino accent of yours truly on the mix. There is a playlist[6] with all VSHN.timer videos you might enjoy, if you are into binge watching, that is.

Writing VSHN.timer every week is a great way to learn about the world of cloud technologies. I can't help but to think about the time when I started working as a software engineer in 1997, writing Active Server Pages apps in VBScript, running them in an absolutely non-virtual Windows NT server.

[1] https://www.vshn.ch/en/blog/vshn-timer

[2] https://vshn.ch

[3] https://share.hsforms.com/1HAWeyoUvRBGoa91MkJnrVg48awa

[4] https://www.tldrnewsletter.com/

[5] https://vshn.tv

[6] https://www.youtube.com/watch?v=1hIzzG88B4E&list=PLms-chRSi2s1uiExml5vMEK1gE0QpZM9r

For good or worse, the technology stack is so much more sofisticated now, and of course the productivity and stability of the tools so much better. But not all is perfect. VSHN.timer is a compendium of the best (and sometimes the worst too) the industry has to offer; we keep an eye on what is coming, what is hot, and what is not, and condense all of it in the shortest format we could find.

And last but not least, we have a blast and we keep learning a lot, which is at least in my case one of my major goals.

So please subscribe, like, and hit that notification button. See you every Monday!

Update, 2021-03-15: Sadly the video version is no longer in production, but you can always subscribe to our RSS feed[7] or to the email newsletter[8].

[7] https://www.vshn.ch/vshn-timer-rss.xml
[8] https://www.vshn.ch/en/newsletter/

The Next Big Thing

Adrian Kosmaczewski

2020-11-21

Looking backwards, the migration from Objective-C to Swift as main programming language for the Apple galaxy was quite an event.

Thanks to the iPhone Objective-C rose from absolute obscurity to be (apparently[1]) the most popular language in the world. Before this happened, the biggest "influencer" for the programming language was, without a hint of a doubt, Scott Stevenson. Two resources he created definitely showed the way for the rest of us. Cocoa Dev Central[2] and his own blog, Theocacao[3].

From the former, suffice to mention gems like the "Learn Cocoa[4]" tutorial. Delightfully crafted, it stands out even today as a superb example of craft and dedication. Thankfully some online resources these days have such attention to detail; Scott set a style and a pace.

I learnt Objective-C reading Duncan Davidson's excellent "Learning Cocoa with Objective-C[5]" book; I then read the massive "Cocoa Programming[6]" volume. Those were the basis of my learning.

The rise of the iPhone brought a new wave of Objective-C experts to the spotlight. Some names that come to mind are Jeff LaMarche and Dave Mark, whose book "Beginning iPhone Development[7]" was actually the first to hit the shelves after Apple dropped the ~~fucking~~ ignominous NDA around the iPhone SDK in October 2008.

I must mention other close friends of mine, namely Daniel Steinberg[8] and Graham Lee[9] from whom I have learnt a lot over the years.

And then, one day in June 2014, Apple surprised everyone (apparently, even their own employees) by announcing a brand new programming language for their platform. A notorious programming lan-

[1] https://www.tiobe.com/tiobe-index/
[2] http://www.cocoadevcentral.com/
[3] http://theocacao.com/
[4] http://www.cocoadevcentral.com/d/learn_cocoa/
[5] https://www.oreilly.com/library/view/learning-cocoa-with/0596003013/
[6] https://www.oreilly.com/library/view/cocoa-programming/0672322307/
[7] https://www.apress.com/gp/book/9781430216261
[8] https://dimsumthinking.com/
[9] https://www.sicpers.info/

guage known as Swift[10].

The change was dramatic. From night to day, a whole new set of "influencers" took the relay and started pouring out content. Probably the biggest supernovas in this new galaxy are right now Paul Hudson[11] and John Sundell[12].

Being a multi-paradigm language, the blogosphere and the podcastsphere had an endless amount of stuff to rewrite and teach in Swift: design patterns, currying, code with emojis, custom operators, obscure generics tricks, data structures and algorithms taken from the SICP book, anything was good to show.

Once again.

I can think of one example of a "survivor" in this galactic collision; Daniel Steinberg, who has a long story of adapting himself to technology change. After having published books about Java in the 1990s, he wrote about Objective-C in the 2000s, about the iPad in the early 2010s, and now about Swift in the 2020s. I have tremendous respect for him.

As far as I am concerned, I never really liked Swift, and I missed Objective-C a lot. There was not a lot left for me to do than get out of that galaxy, which I did.

I actually liked Objective-C because messages were late bound and it had a quite accomodating type system. Not too strict, not too lazy. It compiled apps extremely fast (in any case faster than Swift) and those were precisely the reasons why it was such an awesome thing to my eyes. And it was perfectly compatible with C. Throw any C library to it, and done, you could reuse the knowledge and wisdom of almost 45 years of C programming.

45 years. Our industry is barely 60 years old, yet we could reuse 45 years worth of knowledge.

Oh, but it has pointers and square brackets. Yuck.

I would have loved for Swift to be more compatible with Objective-C, in a way similar to what I saw between Kotlin and Java. The relationship between Swift and Objective-C was conflictive at best, which led to a lot of wheel reinventions over the years.

Kotlin automatically reuses any jar file compiled with Java 1.1 in 1998. Swift reusing Objective-C code? Good luck with those bridging headers.

I did my share of wrapping C libraries into an Objective-C component that would be ultimately consumed by Swift. I've had enough of that. Hopefully it's got better in the past two years, but I can't say.

[10] https://swift.org/

[11] https://www.hackingwithswift.com/

[12] https://www.swiftbysundell.com/

In my eyes, the lack of respect of the Swift language towards the legacy of Objective-C got reflected in the community. I have not seen such disregard for Java in the Kotlin community, or in any other JVM language, for that matter.

The fact that the Swift announcement caught Apple employees off-guard was evident since day one. The tooling around Swift took years to reach a relatively usable level of stability and productivity, and by that time I was already looking somewhere else. Even Visual Studio Code has better refactoring for TypeScript code than Xcode has ever had for Swift.

The power and usefulness of a programming language is not centered just around the language itself; it is also (I'd say mostly) the community and the tooling built on top that bring value to the ecosystem. Where were the linters, the documentation generators, the rock-solid IDEs, the ABIs, the compatibility layers, that would have made Swift a productive tool? Nowhere to be seen.

Apple didn't pay a dime for the countless failed projects trying to fit Swift 1 and 2 into production apps; they used the community as beta testers for at least two years, then they threw the towel and "open sourced" it. And now they want their 15% out of every penny you do on the App Store. And they still ship a failed IDE, making alternatives if not impossible, at least downright difficult to use.

And people still defend them. What is this world we're living in? What kind of madness is this?

But sure, Swift has a relatively neat syntax, and people like it a lot. Well, to me it looks mostly like any language from the 2010s; just take a look at Scala, Rust, Go, or Kotlin. You heard it: nothing to phone home about.

Of course those pundits will scream at me of how the philosophy and the runtime and whatnot is different. The truth is, those languages have very solid ecosystems built around them; the ones around Go and Scala are outstandingly big and make developers immensely productive.

The immaturity of our craft is never more visible than in the waves of new generations of programmers spitting on the work of their slightly older peers, bleeding their "two cents" and pitiful hubris on online forums. Throwing the work of past generations to the same pit where women, people of color, or any programmer using their "nemesis" programming language, must go and die under the rocks of shame and collective laughter.

All of this cacophony is supported by an entire industry of startup founders ready to hire 18 year old "ninjas" cranking code for 80 hours a week for dwingling salaries, and by a legion of clueless recruiters asking for 10 years of experience in technologies released a few months ago.

The lack of respect of otherwise supposedly professional developers towards those that came before them (even when that was just five years ago) is outstanding, appalling, pathetic, disrespectful, moronic, and brings a shadow of morbidity upon our craft.

Bracing for the next wave of “great things” that await us in the 2020s.

We move towards a world with such big social, economic and political issues, that they will make the Y2038 problem[13] seem more irrelevant than the colors on the keycaps of your clicky keyboard. The same keyboard you should raise your eyes from, open a window, and look at the real world from time to time.

[13] https://en.wikipedia.org/wiki/Year_2038_problem

Somebody Call an Asciidoctor

Adrian Kosmaczewski

2020-11-27

Markdown is a great thing; but nothing beats Asciidoc for complex documentation projects.

I discovered Asciidoc while writing the books I published at O'Reilly. Back in 2012 they had an astonishing publishing pipeline built around it, following an early form of "GitOps". Or, rather, "SubversionOps": commit your changes to the Subversion repository, and a few minutes later download a fully formatted PDF file, including all the fonts, looking exactly as the real thing.

Until around a decade ago though, Asciidoc had a bit of a problem: the only available implementation[1] back then was an aging project built in Python somewhere in the 2000s, and it seemed kind of stuck. Dan Allen thought the same, and started a major undertaking: thus was born the Asciidoctor[2] project.

Nowadays, Asciidoctor is a beast of a project, with implementations in Ruby and JavaScript[3], including extensions for EPUB[4] and PDF[5] generation, diagramming[6], support for source code[7] and tables[8], mathematics[9], presentation slides[10], user interface macros[11] for menu entries and buttons, and much, much more. It is a serious contender to the title of the most awesome technical documentation tool ever created.

Asciidoctor is more suitable for large documentation projects thanks to the `include` directive[12], allowing large documents to be split in smaller ones; a painfully missed option in Markdown, in my opinion, which is an otherwise brilliant and ubiquitous format.

[1] https://asciidoc.org/
[2] https://asciidoctor.org/
[3] https://asciidoctor.org/docs/asciidoctor.js/
[4] https://asciidoctor.org/docs/asciidoctor-epub3/
[5] https://asciidoctor.org/docs/asciidoctor-pdf/
[6] https://asciidoctor.org/docs/asciidoctor-diagram/
[7] https://asciidoctor.org/docs/user-manual/#source-code-blocks
[8] https://asciidoctor.org/docs/user-manual/#tables
[9] https://github.com/asciidoctor/asciidoctor-mathematical
[10] https://asciidoctor.org/docs/asciidoctor-revealjs/
[11] https://asciidoctor.org/docs/user-manual/#user-interface-macros
[12] https://asciidoctor.org/docs/user-manual/#include-directive

And if all of that was not enough, Asciidoctor begat Antora[13], a complete website generator with all the bells and whistles one would expect in 2020.

(Actually, Antora is not the only website generator that supports Asciidoc; Hugo[14] does it off-the-box as well.)

Antora has a few features that set it completely apart in its market:

- Complete separation from content and form; we have created various different Antora themes, for example for VSHN[15], Project Syn[16], and APPUiO[17], and we can choose any of these for our websites. We even have a Cookiecutter[18] that offers users the choice of which theme to use when creating a new Antora project.
- The ability to fetch and coalesce documentation sets from various different websites, generating a single website with all of that documentation put together. This is how we created the Project Syn[19] documentation, merging the docs from its different components: Commodore[20], Lieutenant API[21] and more.

Being a static site generator, Antora does not provide a built-in search engine system; fear not, we have developed our own embedded search engine[22] with its dedicated indexer[23], all collaborating together nicely in our GitLab CI/CD pipeline.

Not only that! We have created an Antora Preview[24] container image, used by documentation authors to `make preview` their docs on a browser in real time.

I plead guilty, your honor, and without regrets I affirm I am one of the principal instigators of the introduction of Asciidoctor and Antora at VSHN, in all of its forms. At this moment we are actively using it to generate documentation websites such as the Handbook[25], the Knowledge Base[26], and the Project Syn documentation[27]. We use it also to create presentation slides ready to be deployed in APPUiO[28], our container platform based on OpenShift.

[13] https://antora.org/
[14] https://gohugo.io/
[15] https://github.com/vshn/antora-ui-default
[16] https://github.com/projectsyn/antora-ui-default/
[17] https://github.com/appuio/antora-ui-default/
[18] https://pypi.org/project/cookiecutter/
[19] https://docs.syn.tools/
[20] https://github.com/projectsyn/commodore
[21] https://github.com/projectsyn/lieutenant-api
[22] https://github.com/vshn/embedded-search-engine
[23] https://github.com/vshn/antora-indexer-cli/
[24] https://github.com/vshn/antora-preview
[25] https://handbook.vshn.ch/
[26] https://kb.vshn.ch/
[27] https://docs.syn.tools/syn/index.html
[28] https://appuio.ch/

Thanks to Asciidoctor's JavaScript API we even built a document generation application, where we use Asciidoc templates to generate PDF, Word, and LibreOffice documents, integrating the whole thing into SignRequest, speeding up the work of our sales team. This app is one of our secret weapons to serve our customers better.

The choice of an open standard such as Asciidoc, with a lively and powerful implementation such as Asciidoctor, in turn powering a documentation workhorse such as Antora, has transformed our documentation workflow and keeps on offering new possibilities. I can heartily recommend them all.

Search Engine for akos.ma

Adrian Kosmaczewski

2020-12-04

Adding a search engine to this website was a nice little weekend project.

I build the static HTML website you're reading now using Hugo[1]. Of course, Hugo being what it is, it is clear that there is no server-side process to handle a dynamic request, like a search query would need to be processed. The whole website consists of rather inert HTML files.

This blog post will describe the idea and the process that led to the search box that you can see in the menu of this website.

Design

I host this website in Hostpoint[2], a well-known hosting company from Zürich, Switzerland. They offer the usual "PHP + MySQL" hosting combo, coupled with a lot of goodies, like SSH access and cron jobs, the latest PHP version, all nicely located within FreeBSD servers. I like them very much.

(Of course I should be hosting these pages in APPUiO[3] instead, but let's say for historical reasons I'm still with Hostpoint. I will move to APPUiO at some point.)

All of this meant that the easiest path to have a search engine in this website was to create a PHP 7.4 application returning search results, somehow. After a bit of research I found TNTSearch[4], a fully featured full text search engine written in PHP.

And that was exactly what I needed.

Indexing the Contents

The following challenge was to create a small application that uses TNTSearch to create a suitable index out of the HTML pages generated

[1] https://gohugo.io/
[2] https://www.hostpoint.ch/en/
[3] https://appuio.ch/
[4] https://github.com/teamtnt/tntsearch

by Hugo.

TNTSearch stores the index in a PDO-compatible database; usually MySQL or SQLite[5]. I chose the latter since it is extremely simple to deploy, and I could rebuild and `scp` the new index after publishing a new article every weekend.

I created a new PHP project using Composer[6]. I specified TNTSearch as the only requirement, and then created a small PHP script to generate the index, to be used in the command line.

```php
<?php
$tnt = new TeamTNT\TNTSearch\TNTSearch;
$config = [
    'storage'   => __DIR__,
    'driver'    => 'filesystem',
    'location'  => $docs_root,
    'extension' => 'html',
    'exclude'   => $files_to_exclude
];

// Perform the indexing
$tnt->loadConfig($config);
$indexer = $tnt->createIndex('search/index.db');
$indexer->run();
```

Run this script using the usual `php create_index.php` and you end up with a SQLite 3 file called `index.db`.

Making the Search Results Nicer

A quick analysis of the final `index.db` file (for example using DB Browser for SQLite[7]) shows that the indexing process only stores list of words, and their associations to a certain HTML page, referenced by its absolute path. In order to have nice search results, we need to have a way to show the user at least the title and maybe even a snippet of each search result.

For that, I switched to Python, because of Beautiful Soup[8], a fantastic library that reads HTML and offers a nice API to find out various pieces of information.

So here is how I used it:

```
from bs4 import BeautifulSoup
import sqlite3

def page_title(path):
```

[5] https://sqlite.org/index.html
[6] https://getcomposer.org/
[7] https://sqlitebrowser.org/
[8] https://www.crummy.com/software/BeautifulSoup/

```
    f = open(path, "r")
    html = f.read()
    soup = BeautifulSoup(html, 'html.parser')
    title = soup.find('title')
    return title.string.replace(' | akos.ma', '')

def read_index(index):
    conn = sqlite3.connect(index)
    c = conn.cursor()
    data = []
    for row in c.execute('SELECT id, path FROM filemap'):
        path = row[1]
        title = page_title(path)
        snippet = page_first_paragraph(path)
        data.append({
            'path': path,
            'title': title,
            'snippet': snippet
        })
    conn.close()
    return data

def write_db(filename, data):
    conn = sqlite3.connect(filename)
    c = conn.cursor()
    c.execute('CREATE TABLE IF NOT EXISTS files(id INTEGER PRIMARY KEY, path TE
    c.execute('CREATE INDEX paths ON files(path);')
    for row in data:
        params = (row['path'], row['title'], row['snippet'],)
        c.execute('INSERT INTO files (path, title, snippet) VALUES (?, ?, ?);',
    conn.commit()
    conn.close()

data = read_index('search/index.db')
write_db('search/files.db', data)
```

And now we have two SQLite files: `index.db` with the TNTSearch index, and `files.db` with a list of files including their title, a snippet, and an index for querying stuff using the file path.

Searching

And now we can write the actual PHP application which, once installed in our server, will return a JSON string with the search results:

```
<?php
$query = $_GET['q'] ?? '';
$response = [
    'results' => []
];
```

```php
if ($query === '')
{
    echo(json_encode($response));
}
else
{
    $tnt = new TNTSearch;
    $tnt->loadConfig([
        'storage'   => __DIR__,
        'driver'    => 'filesystem',
    ]);
    $tnt->selectIndex('index.db');
    $tnt->asYouType = true;

    $search_results = $tnt->search($query, 10);

    $files_db_path = realpath(__DIR__ . '/files.db');
    $access = [PDO::SQLITE_ATTR_OPEN_FLAGS => PDO::SQLITE_OPEN_READONLY];
    $db = new PDO('sqlite:' . $files_db_path, null, null, $access);
    $select = $db->prepare('SELECT title, snippet FROM files WHERE path = :path

    foreach ($search_results as $key => $value)
    {
        $path = $value['path'];
        $select->bindParam(':path', $path, PDO::PARAM_STR);
        $select->execute();
        $result = $select->fetch();
        $title = $result['title'];
        $snippet = $result['snippet'];
        $response['results'][] = [
            'path' => short_version($path),
            'title' => $title,
            'snippet' => $snippet
        ];
    }
    header('Content-Type: application/json');
    echo(json_encode($response));
}
```

This script provides the backend functionality we need in our website.

User Interface

And now for the frontend part. This site uses a modified version of the Noteworthy[9] theme for Hugo. Apart from adding an HTML <input> field to the page and some CSS, the most interesting part is, of course, the JavaScript that handles the searches, itself based on the Vanilla

[9] https://github.com/kimcc/hugo-theme-noteworthy

JS[10] framework, the best there is:

```
; (function () {
  'use strict'

  // Creates the DOM structure of a single search result item
  // The website variable contains the current domain where this code is runnin
  function createSearchResultsDiv (item, website) {
    var searchParagraph = document.createElement('p')
    searchParagraph.className = 'search-paragraph'

    var searchEntry = document.createElement('a')
    searchEntry.innerText = item.title
    searchEntry.href = item.path
    searchEntry.className = 'search-entry'
    searchParagraph.appendChild(searchEntry)

    // ...

    var searchDiv = document.createElement('div')
    searchDiv.className = 'search-div paragraph'
    searchDiv.onclick = function () {
      window.location.href = item.path
    }
    searchDiv.appendChild(searchParagraph)
    return searchDiv
  }

  // Builds the HTML structure of the list of search results
  // The results variable is an array of objects with 'name', 'href' and 'excer
  // The query variable is a string entered by the user.
  function display (results, query) {
    if (isEmptyOrBlank(query)) {
      // Display the original page in lieu of the search results if not done ye
      if (!mainArticle.parentNode) {
        body.replaceChild(mainArticle, searchArticle)
      }
      return
    }
    // Rebuild the contents of the "search results" page
    removeAllChildren(searchArticle)
    var searchTitle = document.createElement('h1')
    searchTitle.className = 'page'
    searchArticle.appendChild(searchTitle)
    searchTitle.innerText = 'Search Results for "' + query.trim() + '"'
    if (results.length === 0) {
      var searchResult = document.createElement('p')
      searchResult.innerText = 'No results found.'
      searchArticle.appendChild(searchResult)
```

[10] http://vanilla-js.com/

```
  } else {
    results.forEach(function (item, idx) {
      var searchDiv = createSearchResultsDiv(item, website)
      searchArticle.appendChild(searchDiv)
    })
  }
  // Replace the current page with a "search results" page if not done yet
  if (!searchArticle.parentNode) {
    body.replaceChild(searchArticle, mainArticle)
  }
}

// Performs the actual search
function search (query, callback) {
  if (isEmptyOrBlank(query)) {
    // Display the original page in lieu of the search results if not done ye
    if (!mainArticle.parentNode) {
      body.replaceChild(mainArticle, searchArticle)
    }
    return
  }
  var XMLHttpRequest = window.XMLHttpRequest
  var xmlhttp = new XMLHttpRequest()

  xmlhttp.onreadystatechange = function () {
    if (xmlhttp.readyState === XMLHttpRequest.DONE) {
      if (xmlhttp.status === 200) {
        callback(JSON.parse(xmlhttp.responseText)['results'])
      } else {
        console.log('Status received: ' + xmlhttp.status)
      }
    }
  }

  var url = '/search/?q=' + encodeURIComponent(query)
  xmlhttp.open('GET', url, true)
  xmlhttp.send()
}

var searchInput = document.querySelector('#search-input')
// ...

var timeout = null
function triggerSearch() {
  if (timeout) clearTimeout(timeout)
  timeout = setTimeout(() => {
    var query = searchInput.value
    search(query, function (results) {
      display(results, query)
    })
```

```
    }, 500)
  }

  // Event to be fired everytime the user presses a key
  searchInput.onkeyup = function () {
    triggerSearch()
  }
})()
```

So every time a user types on the field, we wait for half a second after the last character, and send the search to the server. The results are embedded on the DOM of the same page, and if the user cleans the search field, the original page is displayed instead.

I borrowed this client-side code from the VSHN Antora UI Default[11] project, used to build various Antora[12] documentation websites for VSHN.

Conclusion

It was a nice fun project. I reused stuff I already had done in the past; not only the client-side code, but also the approach of a two-stage search engine: first, the creation of the index, and then its use.

This is the approach I used for the search functionality in the Antora websites we have in VSHN, and you can see it in the Antora indexer CLI[13] project. In that case though, it is written in TypeScript[14], and uses Lunr.js[15] as engine. The index is then used by the embedded search engine[16], deployed as a sidecar pod in the Kubernetes deployment.

I have added a few `Makefile` here and there, so now I can simply `make index` and `make deploy` every time I update this website with new content. The search results are snappy, fast, and relevant. And everybody is happy.

For the future I plan on changing the backend of the search index, and move it to MySQL instead. For the moment the current approach works, and anyway, I do not have huge amounts of traffic in this website (at least not so far).

Bonustrack: Search Engine Libraries

As a bonus, please find below a list of search engine libraries I have found online while searching for options. Maybe you will find this list

[11] https://github.com/vshn/antora-ui-default
[12] https://antora.org
[13] https://github.com/vshn/antora-indexer-cli/
[14] https://www.typescriptlang.org/
[15] https://lunrjs.com/
[16] https://github.com/vshn/embedded-search-engine

useful for your own needs.

Rust

- Tantivy[17] is a full-text search engine library inspired by Apache Lucene and written in Rust.
- Toshi[18] is meant to be a full-text search engine similar to Elasticsearch. Toshi strives to be to Elasticsearch what Tantivy is to Lucene. Written in Rust.
- Sonic[19] is a fast, lightweight and schema-less search backend, written in Rust.
- Bayard[20] is a full text search and indexing server, written in Rust, built on top of Tantivy.
- MeiliSearch[21] Ultra relevant, instant, and typo-tolerant full-text search API.

Go

- Bleve[22] is a modern text indexing library in Go.
- Blast[23] is a full text search and indexing server, written in Go, built on top of Bleve.
- Riot[24] is Go Open Source, Distributed, Simple and efficient full text search engine.

Python

- Whoosh[25] is a fast, pure Python search engine library.

JavaScript

- Lunr[26], whose description says it all: "A bit like Solr, but much smaller and not as bright". I used this library in the search engine of VSHN Antora websites, such as our Handbook[27] or Project Syn[28] and it also uses an index, but in this case composed of JSON files.

[17] https://github.com/tantivy-search/tantivy
[18] https://github.com/toshi-search/Toshi
[19] https://github.com/valeriansaliou/sonic
[20] https://github.com/bayard-search/bayard
[21] https://github.com/meilisearch/MeiliSearch
[22] https://blevesearch.com/
[23] https://github.com/mosuka/blast
[24] https://github.com/go-ego/riot
[25] https://github.com/whoosh-community/whoosh
[26] https://lunrjs.com/
[27] https://handbook.vshn.ch/
[28] https://docs.syn.tools/

Update, 2022-08-05: Lyra[29] is a fast, typo-tolerant, full-text search engine written in TypeScript.

Others

- Apache Lucene[30] is a high-performance, full-featured text search engine library written entirely in Java.
- Apache Solr[31] is highly reliable, scalable and fault tolerant, providing distributed indexing, replication and load-balanced querying, automated failover and recovery, centralized configuration and more.
- Elasticsearch[32] is a distributed, open source search and analytics engine for all types of data, including textual, numerical, geospatial, structured, and unstructured.

Update, 2022-08-05: FlexSearch[33] is a search engine for .NET applications.

[29] https://lyrajs.io/
[30] http://lucene.apache.org/
[31] http://lucene.apache.org/solr/
[32] https://www.elastic.co/products/elasticsearch
[33] https://flexsearch.net/

Starting a Typescript CLI Project from Scratch

Adrian Kosmaczewski

2020-12-11

The JavaScript ecosystem has grown dramatically in the past decade. It has become so complex, that I've seen many new developers interested in the subject struggle to find out where to start.

I decided to publish a blog post with the required steps to specifically bootstrap a command-line project with TypeScript. This provides a simple, straightforward, very opinionated, tried and tested approach to create a new project with many features required by rational software development processes.

Opinion

I think the tooling around JavaScript has improved a lot lately; it has become a serious contender for building all kinds of applications, and people are using it pretty much for anything and everything these days.

But it remains a complex beast to tame, and in many cases I can hardly recommend all of the frameworks available. Falling in this category I can mention the Ionic Framework[1], for example. Stay away from it if you can. I prefer to focus this blog post into the things that I found actually bring value to projects. These are more related to good practices, like static analysis, testing, documentation, etc.

Having said that, let us state clearly the first factor of success for any JavaScript project: to not use JavaScript at all, but rather TypeScript[2] instead.

TL;DR: Cookiecutter

Because life is too short to read whole blog posts, I've created a Cookiecutter[3] with the final Node.js[4] project; feel free to create your new TypeScript CLI projects from scratch using the following command:

[1] https://ionicframework.com/
[2] https://www.typescriptlang.org/
[3] https://cookiecutter.readthedocs.io/
[4] https://nodejs.org/

```
$ cookiecutter https://gitlab.com/akosma/typescript-cli-cookiecutter
```

If you want to learn how that Cookiecutter was built, keep reading.

Requirements

Make sure you have the following software available before starting:

1. Terminal
2. Visual Studio Code[5]
3. Usual command line tools: `git`, `curl`, etc.

Install Node.js

I never use the standard version of Node.js bundled with my system, if any, nor I install it following the usual advice you find online. Instead I always use nvm[6] to manage my Node.js[7] installations. This way I can have several Node.js versions installed, and I can switch from one to the other with one command. This is particularly useful if you work as a consultant, and you need different versions for different customers.

As an analogy, `nvm` is the equivalent in Node.js to pyenv[8] for Python and rbenv[9] for Ruby.

To install `nvm`, just run the following command:

```
$ curl -o- https://raw.githubusercontent.com/nvm-sh/nvm/v0.37.2/install.sh | ba
```

Then setup your environment in the terminal, adding these lines to your `~/.profile` or `~/.zshrc` file:

```
export NVM_DIR="$([ -z "${XDG_CONFIG_HOME-}" ] && printf %s "${HOME}/.nvm" || p
[ -s "$NVM_DIR/nvm.sh" ] && \. "$NVM_DIR/nvm.sh" # This loads nvm
```

Find the latest versions of Node.js, and install one of them:

```
$ nvm ls-remote | grep "Latest LTS"
$ nvm install 14.15.1
```

I like to install the latest LTS release; please bear in mind that the version numbers will have most probably changed by the time you read this.

Test your current installation with these commands; each output should point to an executable in `~/.nvm/versions/node/v14.15.1/bin`

```
$ which node
$ which npm
$ which npx
```

[5] https://code.visualstudio.com/
[6] https://github.com/nvm-sh/nvm
[7] https://nodejs.org/
[8] https://github.com/pyenv/pyenv
[9] https://github.com/rbenv/rbenv

If you're using Windows, instead of `which` use the `where` command.

We're ready to go now.

New TypeScript Project

Now that we have Node.js installed, we can create a new project:

```
$ mkdir -p project/src
$ cd project
$ npm init
```

At this point you can just accept all defaults. You will find a new file called `package.json` in your project, used by Node.js and `npm` to store information about your project, including dependencies and all metadata.

Let's install TypeScript now:

```
$ npm install typescript ts-node --save-dev
```

This command will create a new folder in your project, the dreaded `node_modules` behemoth, which includes all the dependencies of your project. The `npm install` command has been the source of countless[10] memes[11] in the community.

The `ts-node` project allows you to run TypeScript code on the terminal, compiling it to JavaScript on the fly. This is super useful for CLI apps like the one we're building here.

Now we need to initialize our TypeScript installation with a `tsconfig.json` file:

```
$ npx tsc --init
message TS6071: Successfully created a tsconfig.json file.
```

The `npx` command executes binaries installed in your `./node_modules/.bin` folder. This helps you to avoid having to install commands globally, and will also help later on, in particular for building the application in different environments, such as during the creation of Docker containers.

I recommend to always install all of the tools required for a project locally, and never globally.

You can customize your `tsconfig.json` file, removing all comments and uncommenting the two lines shown below, so that it looks like this:

```
{
  "compilerOptions": {
    "target": "es5",
    "module": "commonjs",
```

[10] https://twitter.com/Mamba_Dev/status/1066015794205782016
[11] https://twitter.com/brad_frost/status/1310973587948531718

```
    "sourceMap": true,  // Uncomment this
    "outDir": "out",    // and uncomment and set this
    "strict": true,
    "esModuleInterop": true,
    "skipLibCheck": true,
    "forceConsistentCasingInFileNames": true
  }
}
```

Write Code

Finally! The interesting part of our application. We are going to use Visual Studio Code to create a class and a program consuming it. I chose Visual Studio Code because it comes from the same people who created TypeScript, and it has stellar support for it.

```
$ code .
```

Create two TypeScript files in the `src` folder; first, `src/customer.ts`:

```
export class Customer {
  readonly name : string

  constructor(name: string) {
    this.name = name
  }

  greet(): string {
      return `Hello ${this.name}`
  }
}
```

This is a very simple TypeScript class, with a constructor and a method called `greet()`. Nothing to phone home about. You will add in this folder your own logic, as required by your own needs.

Then we will create the entry point of our application, `src/index.ts`:

```
import { Customer } from './customer'

const cust = new Customer('Toto')
console.log(cust.greet())
```

You can now test the application by running this command:

```
$ npx ts-node src/index.ts
Hello Toto
```

Tada! Your application is running, greeting through an instance of the `Customer` class.

We can update the `package.json` file now, including some tasks:

```
{
  "name": "project",
```

```
  "version": "1.0.0",
  "description": "",
  "main": "src/index.ts",       // Change this if needed
  "scripts": {
    "build": "npx tsc",         // Add this line
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "",
  "license": "ISC",
  "devDependencies": {
    "ts-node": "^9.0.0",
    "typescript": "^4.0.3"
  }
}
```

The changes above allow us to type the following commands:

```
$ npm run build
$ node out/src/index.js
Hello Toto
```

The entries in the `scripts` section of the `package.json` file allow us to use `npm` as if it were a `make` command.

Clean Task

It is a good practice to have a `clean` task to quickly remove all build products. We could add a line like this to our `package.json` to do that:

```
  "scripts": {
    "build": "npx tsc",
    "release": "npx gulp",
    "clean": "rm -rf out", // this line
    "test": "npx ts-node node_modules/.bin/jasmine spec/*"
  },
```

But the problem is that the `rm -rf` command does not work in Windows (and yes, I care about our poor Windows users). Thankfully there's a simple workaround to this issue:

```
$ npm install rimraf --save-dev
```

Now we can use the `rimraf` command, which works everywhere:

```
  "scripts": {
    "build": "npx tsc",
    "release": "npx gulp",
    "clean": "npx rimraf out", // better!
    "test": "npx ts-node node_modules/.bin/jasmine spec/*"
  },
```

And now we can clean our project simply by doing

```
$ npm run clean
```

Writing TypeScript

I will not explain all of TypeScript in this tutorial, but just provide a cheatsheet of the most important things you need to know about it:

- All JavaScript is valid TypeScript. Not all valid TypeScript is valid JavaScript (yet).
- All JavaScript numbers are floats. There are no integers. Pay attention to rounding errors.
- Use `const` and `let` instead of `var`.
- Use `for of` instead of `for in`.
- Functions are first-class objects.
- String interpolation and multi-line strings use backticks: `Some text and a ${variable}`.
- Forget about semicolons; the compiler will add them.
- Add comments, lots of comments; the compiler will strip them.
- Surround strings with 'single quotes'. This is particularly useful to output strings that contain double quotes, like in HTML.
- Variables and classes in TypeScript "work as expected", unlike regular JavaScript.
- There is a module system similar to that of Python.
- The TypeScript type system is outstanding: type inference, interfaces, classes, enums, generics with constraints, literal types, union and intersection types, nullable types, type aliases, structural types, function decorators…
- There are `@types/xxxxx` packages available through `npm`, which provide type information for many popular JavaScript packages, so that you get a better developer experience. Not all packages have one, but most do.
- Use type any only to interact with existing JavaScript; always type variables and return values explicitly in your own code.
- Prefer inline functions `() => {}` to classic `function` keyword.
- Make numbers more readable with the thousands separator: `1_000_000`.
- All JavaScript runs interpreted and single-threaded, in a single event loop. Multiprocessing is achieved through callbacks.
- Use `await` / `async` and `Promises` instead of "callback hell". This advice also includes using promises in the `fs` package.

Add Unit Tests

Mocha[12] is a popular JavaScript testing library. Chai[13] is an assertion library with various styles (should, assert, expect, etc). The `ts-mocha` project provides a way to directly execute Mocha tests written in TypeScript.

[12] https://mochajs.org/
[13] https://www.chaijs.com/

Install Mocha and Chai in your project, including the TypeScript type information for each:

```
$ npm install mocha @types/mocha ts-mocha chai @types/chai --save-dev
```

Create a file called `spec/customer.ts` to hold the tests:

```
import { expect } from 'chai'
import { Customer } from '../src/customer'

describe('A Customer', () => {
  it('greets', () => {
    const cust = new Customer('Toto')
    expect(cust.greet()).to.equal('Hello Toto')
  })
})
```

Modify `package.json` to add a test command

```
  "scripts": {
    "build": "npx tsc",
    "clean": "rimraf out",
    "test": "npx ts-mocha spec/*" // Add this line
  },
```

Run `npm test` or `npm run test`:

```
> project@1.0.0 test /home/akosma/Dropbox/Current/typescript-project
> npx ts-mocha spec/*

  A Customer
    ✓ greets

  1 passing (3ms)
```

For convenience, I suggest adding the Mocha Test Explorer[14] in Visual Studio Code to have integrated testing support. This extensions uses another one[15] from the same author, which works with many other programming languages.

By the way, if you are a fan of JetBrains IDEs, you should know that Mocha is also natively supported in WebStorm.

Git

At this point we should add this project to source control. Create a simple `.gitignore` file with the following contents

```
node_modules
out
```

[14] https://marketplace.visualstudio.com/items?itemName=hbenl.vscode-mocha-test-adapter

[15] https://marketplace.visualstudio.com/items?itemName=hbenl.vscode-test-explorer

```
dist
```

Run the usual commands to store everything properly:

```
$ git init
$ git add .
$ git commit -m "First commit"
```

Much better.

Gulp

Gulp[16] is a very common JavaScript build tool. It reads tasks from a `gulpfile.js`, just like `make` would use a `Makefile`, or `ant` would use its `build.xml`.

This `gulpfile.js` is intended for command-line apps; for client web apps, one must use other libraries, such as `browserify` and `tsify`, which require a slightly different setup.

Bear in mind that this `gulpfile.js` provides uglification. This not only provides some obfuscation of your code, it actually helps bootstrapping the code faster in the runtime.

Install the required dependencies:

```
$ npm install gulp gulp-typescript gulp-uglify gulp-chmod gulp-insert @types/no
```

Create `gulpfile.js` at the root of the project:

```
var gulp = require('gulp')
var ts = require('gulp-typescript')
var uglify = require('gulp-uglify')
var chmod = require('gulp-chmod')
var insert = require('gulp-insert')
var tsProject = ts.createProject('tsconfig.json')

function build() {
  return gulp.src('src/*.ts')
    .pipe(tsProject())
    .pipe(uglify())
    .pipe(gulp.dest('dist'))
}

function release() {
  return gulp.src('dist/index.js')
    .pipe(insert.prepend(`#!/usr/bin/env node
`))
    .pipe(chmod(0o755))
    .pipe(gulp.dest('dist'))
}
```

[16] https://gulpjs.com/

```
exports.default = gulp.series(build, release)
```

Pay attention to the line in the `release()` task, with the new line character at the end! That string is surrounded by backticks (“‘”) so that we can use a multiline string.

Modify `package.json` to add a new task, and a new folder to clean at the end:

```
  "scripts": {
    "build": "npx tsc",
    "release": "npx gulp",      // Add this line
    "clean": "rimraf out dist", // Add another folder to clean
    "test": "npx ts-mocha spec/*"
  },
```

Run the new command to drive `gulp` and build the application:

```
$ npm run release
$ dist/index.js
Hello Toto
```

Debug in Visual Studio Code

Instead of adding `console.log()` statements all over the place, we’d better use a debugger. We are going to configure Visual Studio Code for that.

In Code, use the `SHIFT + CTRL + D` shortcut (or its equivalent in the Mac) to open the Debug explorer. Click on the “create a launch.json file” link below the blue “Run and Debug” button of the debug explorer.

Select Node.js in the menu that appears.

Add the `preLaunchTask` element below in the `launch.json` file, and modify the entry point in the `program` element:

```
{
    "version": "0.2.0",
    "configurations": [
        {
            "type": "node",
            "request": "launch",
            "name": "Launch Program",
            "skipFiles": [
                "<node_internals>/**"
            ],
            "program": "${workspaceFolder}/src/index.ts",
            "preLaunchTask": "tsc: build - tsconfig.json",   // Add this line
            "outFiles": [
                "${workspaceFolder}/**/*.js"
            ]
```

```
        }
    ]
}
```

Open `src/index.ts` and place a breakpoint (or use the `debugger` keyword). Hit the F5 key, select "Node.js" in the menu, and watch the breakpoint hit.

If it didn't open automatically, open the View menu, select "Debug Console" (`SHIFT + CTRL + Y`) and see the execution of the code.

Add External Libraries

The JavaScript ecosystem is flooded with libraries, of various degrees of quality. In this section we're just going to add one, and use it in our code.

Install the library locally, this time without the `--save-dev` parameter, as we want this library to be available during runtime, and not just at build time:

```
$ npm install cowsay
```

Execute `npm audit fix` if needed.

Modify `src/customer.ts` to use the library:

```
var cowsay = require('cowsay')

export class Customer {
  readonly name: string

  constructor(name: string) {
    this.name = name
  }

  greet(): string {
    return cowsay.say({ text: `Hello ${this.name}` })
  }
}
```

Test run the code:

```
$ npx ts-node src/index.ts
 _____________
< Hello Toto >
 -------------
        \   ^__^
         \  (oo)\_______
            (__)\       )\/\
                ||----w |
                ||     ||
```

Fix the unit test (remember to use double backslash \\ to escape the single backslash, and make the test pass):

```
import { expect } from 'chai'
import { Customer } from '../src/customer'

describe('A Customer', () => {
  it('greets', () => {
    const cust = new Customer('Toto')
    expect(cust.greet()).to.equal(` _____________
< Hello Toto >
 -------------
        \\   ^__^
         \\  (oo)\\_______
            (__)\\       )\\/\\
                ||----w |
                ||     ||`)
  })
})
```

Run the tests now, they should pass:

```
$ npm test
```

API Documentation

Now that we have a nice application, we need to document its APIs so that other developers can use our code.

Open the `src/customer.ts` file and type `/**` at the beginning of the `greet()` method. Hit ENTER and start writing the API documentation; it does not matter, just write something.

Open `src/index.ts`, and hover the mouse on top of the call to the `greet()` method to see the API docs in use.

You can export the API documentation into a nice HTML website using TypeDoc[17], which is the standard[18] API documentation tool for TypeScript.

```
$ npm install typedoc --save-dev
$ npx typedoc --out docs -theme minimal src
```

Open `docs/index.html` to see the generated documentation.

Add `docs` to `.gitignore`. We don't want to store the generated documentation folder in Git.

```
node_modules
out
dist
docs
```

[17] https://typedoc.org/
[18] https://github.com/Microsoft/tsdoc

Edit `package.json` to add a `scripts` entry:

```
  "scripts": {
    "build": "npx tsc",
    "release": "npx gulp",
    "clean": "rimraf out dist docs", // Add a new folder to clean
    "docs": "npx typedoc --out docs -theme minimal src", // Add this line
    "test": "npx ts-mocha spec/*"
  },
```

ESLint

TypeScript ESLint[19] is a project that provides a linter for TypeScript code. Linters perform static analysis of your code, showing you areas of improvement, and even highlighting potential security flaws. I strongly recommend to use a linter for your project, whatever programming language you use.

Install TypeScript ESLint:

```
$ npm install eslint typescript @typescript-eslint/parser @typescript-eslint/es
```

Create an `.eslintrc.js` file in the root of the project with these contents:

```
module.exports = {
  root: true,
  parser: '@typescript-eslint/parser',
  plugins: [
    '@typescript-eslint',
  ],
  extends: [
    'eslint:recommended',
    'plugin:@typescript-eslint/recommended',
  ],
};
```

Create an `.eslintignore` file to avoid linting many folders in the project:

```
node_modules
out
dist
docs
```

Run the linter:

```
$ npx eslint src

~/project/src/customer.ts
  1:1   error  Unexpected var, use let or const instead        no-var
  1:14  error  Require statement not part of import statement  @typescript-esli
```

[19] https://typescript-eslint.io/

```
✖ 2 problems (2 errors, 0 warnings)
  1 error and 0 warnings potentially fixable with the `--fix` option.
```

Another benefit is that Visual Studio Code automatically detects the presence of ESLint and highlights the issues in our code.

We can run the command with the `--fix` option, which will remove one of the errors:

```
npx eslint src --fix

~/project/src/customer.ts
  1:16  error  Require statement not part of import statement  @typescript-esli

✖ 1 problem (1 error, 0 warnings)
```

The remaining problem has to do the `require` statement; in TypeScript it is preferred to use the `import` statement instead, but the Cowsay library unfortunately does not include a `@types/cowsay` library with the type information. At the time of this writing there's an open pull request[20] waiting to be merged that precisely solves this issue.

In this case we will add a statement in our code to prevent ESLint from complaining about this fact:

```
/* eslint-disable @typescript-eslint/no-var-requires */
const cowsay = require('cowsay')

export class Customer {
  readonly name: string

  /**
   * This is a Customer, the most important thing, ever.
   */
  constructor(name: string) {
    this.name = name
  }

  /**
   * This method makes the customer greet.
   */
  greet(): string {
    return cowsay.say({ text: `Hello ${this.name}` })
  }
}
```

This will silence the linter until you find a better solution, which in this case would involve writing a types definition file[21] for cowsay or

[20] https://github.com/piuccio/cowsay/pull/59

[21] https://www.typescriptlang.org/docs/handbook/declaration-files/introduction.html

waiting for the pull request[22] to be merged.

Finally, as usual, add a command to your `scripts` section of the `package.json` file:

```
  "scripts": {
    "build": "npx tsc",
    "release": "npx gulp",
    "clean": "rimraf out dist docs",
    "docs": "npx typedoc --out docs -theme minimal src",
    "lint": "npx eslint src", // Add this line
    "test": "npx ts-mocha spec/*"
  },
```

Docker Container

Docker containers are very useful ways to encapsulate command-line applications, making it much simpler to integrate in CI/CD build systems, or to distribute to colleagues.

Create a `Dockerfile` for your application

```
# Step 1: Builder image
FROM node:14.15.1-alpine3.12 AS builder
COPY [".eslintrc.js", ".eslintignore", "tsconfig.json", "gulpfile.js", "package
COPY src /command/src
COPY spec /command/spec
WORKDIR /command
RUN npm install
RUN npm audit fix
RUN npm test
RUN npm run lint
RUN npm run release

# After the build, this only installs the libraries used in production,
# not the ones installed with the `--save-dev` parameter
RUN rm -rf node_modules
RUN npm install --production

# Step 2: Runtime image
FROM astefanutti/scratch-node:14.14.0
COPY --from=builder /command/node_modules /app/node_modules
COPY --from=builder /command/dist/*.js /app/
ENTRYPOINT ["node", "/app/index.js"]
```

In step 1 we execute the tests and linter; in case of an error in either case, the build process will stop automatically.

The runtime image used in step 2 is provided by Antonin Stefanutti[23] from Red Hat, and provides a lightweight base image for Node.js ap-

[22] https://github.com/piuccio/cowsay/pull/59
[23] https://github.com/astefanutti

plications. This way our final image is 41 MB, instead of the… 960 MB of the standard Node.js image[24], or the 122 MB of the `node:alpine` image used for the build process in step 1. Quite a difference.

Run the following commands using either Docker[25] or Podman[26]:

```
$ podman build -t typescript-tutorial .
```

Once the image is built, you can run it directly:

```
$ podman run --rm typescript-tutorial
```

And you can see it in your local registry, and remove it if needed:

```
$ podman images | grep typescript-tutorial
$ podman rmi typescript-tutorial
```

From this point on, you can share this image with others through a registry, like Docker Hub[27], Quay[28], or private registries such as Harbor[29], GitLab[30], or OpenShift[31].

What Next?

Here we are. We have a fully functioning TypeScript CLI project, ready to receive your own code and tests. It can be statically analyzed, built as an OCI container image, and a size-optimized one for that matter.

What else can you do now with your project? Although I've kept the examples and workflows as simple as possible in this text, there's a lot of extension points for your application:

1. Use the pkg[32] module to encapsulate code in a portable executable, ready to be distributed without fear of dependencies.
2. Expand `.gitignore` with https://gitignore.io/api/node
3. Configure WebStorm[33] to work with this project (and adapt `.gitignore` accordingly).
4. Add logging with winston[34].
5. Add command-line argument parsing with commander[35] or with oclif[36].
6. Use `Promise` and `await` / `async` for asynchronous code.

[24] https://hub.docker.com/_/node/
[25] https://www.docker.com/
[26] https://podman.io/
[27] https://hub.docker.com/
[28] https://quay.io/
[29] https://goharbor.io/
[30] https://docs.gitlab.com/ee/user/packages/container_registry/
[31] https://docs.openshift.com/container-platform/4.6/registry/architecture-component-imageregistry.html
[32] https://www.npmjs.com/package/pkg
[33] https://www.jetbrains.com/webstorm/
[34] https://www.npmjs.com/package/winston
[35] https://www.npmjs.com/package/commander
[36] https://oclif.io/

7. Strip `alert()` and `console.log()` calls at build time with gulp-strip-debug[37].
8. Create a desktop application and distribute it with Electron[38].
9. Manipulate date and time information with Moment.js[39]
10. Run shell commands from your own code using shelljs[40]
11. Create PDFs with PDF-lib[41]
12. Style `console.log()` to your liking[42]

[37] https://github.com/sindresorhus/gulp-strip-debug
[38] https://www.electronjs.org/
[39] https://momentjs.com/
[40] https://www.npmjs.com/package/shelljs
[41] https://pdf-lib.js.org/
[42] https://denic.hashnode.dev/use-consolelog-like-a-pro

Manifestos and Checklists

Adrian Kosmaczewski

2020-12-18

The Agile Manifesto was probably the first, but certainly not the last attempt at creating a web page that provides a whole platform to a certain ideology.

During the past 20 years, lots of people got fed up with some situation, and wrote a web page (sometimes, a whole website) dedicated to scratching that particular itch, usually with the word "Manifesto" on it. Sometimes, instead, using the word "Checklist," but with a similar objective. These articles range from the useful to the funny, and they usually have the best intentions.

This article is a summary of the ones I could find in the past 5 years of collecting such links into a Joplin[1] note that grew uncomfortably big.

Manifestos

Let us start with Manifestos, beginning with the most well known of them all.

Project Management

One does not need to introduce the all-mighty Agile Manifesto[2], which started quite a revolution 20 years ago. In a similar vein we can find the similarly worded and presented Manifesto for Software Craftmanship[3]. Both look quite harmless, from a distance at least.

The implementations of the ideas behind Agile, however, prompted quite a response: the Tarantino-inspired Programming, Motherfucker[4] which is totally not SFW, Artisanal Retro-Futurism crossed with Team-Scale Anarcho-Syndicalism[5] which includes a video, and the Sweary Lightweight Agile Planning or SLAP[6] methodology, which includes a SFW version should you need one.

[1] https://joplinapp.org/
[2] https://agilemanifesto.org/
[3] http://manifesto.softwarecraftsmanship.org/
[4] http://programming-motherfucker.com/
[5] http://arxta.net/
[6] http://slap.pm/

Finally, the level of burnout in our industry due to an endless stream of consecutive scrums, and the increase of teams' "velocity", have brought us the 501 Manifesto[7] and the cry for the 1x Engineer[8].

I hope you can see an evolution in the three paragraphs above, and maybe this will give you an idea of the current state of the software industry.

Update, 2022-09-30: People are asking Apple to Fix Radar or GTFO[9].

Programming

Let us now concentrate in the best part of our craft. You might want to use UTF-8 everywhere[10] in your code and documentation. Whatever language you use, remember to follow the 12 Factor App[11] principles if you are writing backend services, or the 9 iOS Factors[12] in case you are writing apps for the App Store.

If you are into PHP, consider reading PHP the Right Way[13] as well as the Field Guide to Elephpants[14]. Kinda cute. If you are into reactive thingies, which are quite fashionable these days, read the Reactive Manifesto[15] instead.

Of course, writing the code is just a part of the story; another is debugging, and for that, make sure you have a rubber duck[16] nearby.

Once your code runs, you need to release it. For that, use rootless containers[17], follow semantic versioning[18], and keep[19] a changelog[20] (yes, there are two such manifestos about changelogs, clearly they had not heard of one another.)

If your Linux is slow, make it fast again[21].

I assume you are using Git (who is not?) in which case you might be interested in following the principles of GitOps[22] and in reading the Oh Shit, Git!?![23] guide. You are welcome.

[7] https://501manifesto.dev/
[8] https://1x.engineer/
[9] https://fixradarorgtfo.com/
[10] https://utf8everywhere.org/
[11] https://12factor.net/
[12] https://ios-factor.com
[13] https://phptherightway.com/
[14] https://afieldguidetoelephpants.net/
[15] https://www.reactivemanifesto.org/
[16] https://rubberduckdebugging.com/
[17] https://rootlesscontaine.rs/
[18] https://semver.org/
[19] https://keepachangelog.com/
[20] https://changelog.md/
[21] https://make-linux-fast-again.com/
[22] https://www.gitops.tech/
[23] https://ohshitgit.com/

Finally, if you are in the business of writing console (CLI) applications, consider reading the Command Line Interface Guidelines[24] (arguably featuring the best font design of all manifestos in this page) and paying attention to the `DO_NOT_TRACK`[25] variable in your environment.

Update, 2021-02-11: Implement your network protocols without I/O[26].

Update, 2022-05-14: A proposal for self-contained systems[27].

Update, 2022-07-01: The Continuous Architecture manifesto[28].

Update, 2023-09-08: Get better project history through Conventional Commits[29].

Design

There are also manifestos for designers. Let us start with the Laws of UX[30], the Photoshop Etiquette[31] (please rename your layers, people,) and the guide to Brutalist Web Design[32]. I am not a graphic designer, so maybe there are more, but I have not came across any more than these.

Declarations of In(ter)dependence

These manifestos deserve a section of their own. The oldest of these is the EFF Declaration of Cyberspace Independence[33] written in Davos, in February 1996 (when the word “Cyberspace” was still cool). Then comes the Declaration of Interdependence[34] which according to Graham[35] was originally located in Alistair Cockburn[36]’s website.

Finally, here is Larry Sanger’s Declaration of Digital Independence[37].

Whatever these things mean.

---

[24] https://clig.dev
[25] https://consoledonottrack.com/
[26] https://sans-io.readthedocs.io/
[27] https://scs-architecture.org/
[28] https://continuous-architecture.org/docs/manifest/manifesto.html
[29] https://www.conventionalcommits.org/en/v1.0.0/
[30] https://lawsofux.com/
[31] https://photoshopetiquette.com/
[32] https://brutalist-web.design/
[33] https://www.eff.org/cyberspace-independence
[34] https://www.adventureswithagile.com/2014/08/19/declaration-of-interdependence/
[35] https://www.sicpers.info/
[36] https://en.wikipedia.org/wiki/Alistair_Cockburn
[37] https://larrysanger.org/2019/06/declaration-of-digital-independence/

Communication & Business

Sometimes the problems lie in the way people talk to each other, and the following manifestos try to bring solutions to those issues.

Take this one for example: the reason why your question in Stack Overflow receives no answers is because of the way you asked it; check the XY Problem[38] manifesto to learn more and change your question asking habits. It is a bit like a meta-RTFM of sorts.

Or maybe you need to make better comments[39], follow the core protocols[40], or to learn how to explain security honestly[41]. Of course, none of this would matter if you are not working in a responsive organization[42], anyway.

Or you might just need to make clear to somebody who is the team behind a website[43], that is all.

Update, 2022-11-26: During the rise of Mastodon[44], here's a charter for the fediverse[45].

Open Source

The global collaboration required to make good open source software requires lots of goodwill and mutual understanding. For example, if you are just creating a hobby project, make it clear to others that it will go unmaintained[46].

On the other hand, if you want to contribute to open source software, remember to read the Contributor Covenant[47] first.

Maybe you are not a developer, in which case you would like to know what open source software might help you replace some other commercial package, and there is a guide[48] for that as well.

No matter what you want to do with open source software, remember always to first, do no harm[49] and to learn exactly what empowered free software[50] means.

[38] https://xyproblem.info/
[39] https://conventionalcomments.org/
[40] https://thecoreprotocols.org/
[41] https://honest.security/
[42] https://www.responsive.org/manifesto
[43] http://humanstxt.org/
[44] /blog/mastodon
[45] https://slocanstatement.org/
[46] http://unmaintained.tech/
[47] https://www.contributor-covenant.org/
[48] https://switching.software/
[49] https://firstdonoharm.dev/
[50] https://www.sicpers.info/2020/02/empowered-free-software/

Do Not Do This, Do Not Use That

Then we have the family of widely negative manifestos. They range from the obvious, such as Chrome is Bad[51] and do not use Medium[52], to the least obvious, like not using the "Hello" word in chats[53] to list all the things that are wrong with YAML[54].

The truth is, the whole industry has been designed to be defective by design[55].

Update, 2022-08-19: Dear Linux developers, please do not ship[56] work in progress to users.

Update, 2022-08-26: Do not send HTML email, only plain-text email[57].

Pandemic

The 2020 pandemic has changed our relationship to work and society. First of all, stay the fuck home[58]. Then, remember that it's OK[59] if you do not feel well. And also, project managers out there, you must adapt to the new remote working[60] conditions ASAP. Did not you get the memo?

Ethics

The software industry keeps its head buried under the sand, failing to acknowledge the role of ethics in society. For those of us who refuse to ignore the truth, here go some enlightening reads: The Ethical Source Movement[61], the Post-Meritocracy Manifesto[62], and a Contract for the Web[63].

Read them. Breathe. Change yourself.

Update, 2022-11-04: The IndieWeb Manifesto[64], more relevant than ever at a time when platforms start going crazy[65].

[51] https://chromeisbad.com/
[52] https://nomedium.dev/
[53] http://www.nohello.com/
[54] https://noyaml.com/
[55] https://www.defectivebydesign.org/
[56] https://do-not-ship.it/
[57] https://useplaintext.email/
[58] https://staythefuckhome.com/
[59] https://its-ok.clearleft.com/
[60] https://www.emergencyremote.com/
[61] https://ethicalsource.dev/
[62] https://postmeritocracy.org/
[63] https://contractfortheweb.org/
[64] https://indieweb.org/
[65] /blog/mastodon/

Economics

We live in the most challenging times of our species. Our relationship with one another and our economic organization must change, and here are two interesting manifestos for that: Common Futures[66] and Participatory Economics[67]. If you give a shit about humanity, you will find them truly inspiring.

Update, 2021-02-08: Here goes a manifesto to avoid having free plans in your SaaS[68].

Update, 2021-04-09: If you are into accounting, you might want to manage your accounts in plain text[69]. Or not.

Checklists

To finish, some checklists you might find useful.

If you deploy in Kubernetes, check the production best practices[70] and the production checklist[71].

With all of those deployments in the cloud, security has become an actual concern. For that, here are the security checklist[72], the Azure checklist[73], and if you use AWS, here is the AWS Security Checklist[74] for you. If you follow DevSecOps practices, here is the DevSecOps Checklist[75].

Conclusion

There is no conclusion, this page is just a list of links, click on them and have fun. I like collecting stuff like that, so expect more posts similar to this one in the future.

[66] https://common-futures.org/index-en.html
[67] https://www.participatoryeconomics.info/
[68] https://nofreeplan.com/
[69] https://plaintextaccounting.org/
[70] https://learnk8s.io/production-best-practices
[71] https://srcco.de/posts/web-service-on-kubernetes-production-checklist-2019.html
[72] https://securitycheckli.st
[73] https://azurechecklist.com/
[74] https://d1.awsstatic.com/whitepapers/Security/AWS_Security_Checklist.pdf
[75] https://www.sans.org/security-resources/posters/appsec/secure-devops-toolchain-swat-checklist-60

How to Desperately Suck at Cliches

Adrian Kosmaczewski

2020-12-25

I was born in Argentina. This fact, all by itself, provides a rather unlimited amount of smiles in every person I meet.

The name "Argentina" invariably evokes ideas of roughness, wilderness, passion, like few countries in the planet. People dream about Argentina. People dream of going there and hitchhike, ride horses with the Gauchos in the Pampas[1], following the steps of Che Guevara. People picture themselves attending a Superclásico[2], screaming[3] with another eighty thousand people as the ball reaches the net. People dream of colors, smells and sounds yet unforeseen, offered only to the pleasure of a lucky few. Maradona. The Pope. Evita. Borges. Messi. All together, around a gigantic barbecue.

But wait, it gets better (or worse.) For reasons that go beyond the scope of this text, I happen to have lived in Switzerland for almost 30 years. Which, in itself, is another interesting fact to talk about in any social situation.

Ah, Switzerland, they say. Mountains. Watches. Punctuality. Trains. Incredible landscapes. James Bond movies. Cheese. Lakes. Peace conventions. Humanitarian organisations. Banks. Cheese fondues. Victorinox knives. Neutrality. Ski resorts. More cheese. The Matterhorn[4] on a postcard, with the soothing sound of the Alphorns[5] in the background.

Argentina and Switzerland. Switzerland and Argentina. Both countries with amazing clichés, deeply rooted in the minds of people all over the world. Go anywhere in the world, mention those two countries, and somebody will start dreaming for sure. Guaranteed.

As much as I love these, my beloved two countries of citizenship, the truth is that I suck at being both Argentine and Swiss.

Seriously. I mean it. Deeply. Almost to the point of shame.

Let me explain with an enumeration of utter failures:

[1] https://en.wikipedia.org/wiki/Pampas
[2] https://en.wikipedia.org/wiki/Supercl%C3%A1sico
[3] https://www.youtube.com/watch?v=jqgG5WUsXQA
[4] https://en.wikipedia.org/wiki/Matterhorn
[5] https://en.wikipedia.org/wiki/Alphorn

- I do not dance tango.
- I have never ridden a horse. Ever.
- I do not play football (also known as soccer by my American readers.) I tried. Better not say more.
- I do not particularly enjoy going to the mountains – needless to say, I do not ski, either.
- I do not even know how to play Jass[6]. I do play Truco[7], though. I used to be quite good at it even.
- I prepare a rather unremarkable maté[8].
- My travels through the geography of each country has been so far limited to a few big cities.
- My Argentine friends always tell me that I am too polite, punctual and organised.
- My Swiss friends tell me I am too disorganised and blunt.
- Oh, but I did my military service as a soldier of the legendary Swiss Army. A rather awkward club to belong to, if you ask me.

In any case I am a voracious meat and cheese eater. I am an epicurean deeply in love with the culinary gifts offered by both countries, but, needless to say, I am not a very good cook of either.

TL; DR: I am too Swiss to be Argentine, and too Argentine to be Swiss.

As the reader can imagine, at some point I came to the conclusion that none of these labels really matter. People are very keen to fit everything and everybody in little boxes, in order to understand and to manage. In my case, none of that crap ever works. I do not fit any box, and I do not want to fit in any.

And neither should anyone try to fit in anyone else's boxes, and neither should anyone try to fit other people in their own little boxes. Just let everyone be what they are.

Oh and, by the way, yes, my family name is Polish. Do not get me started on that one.

[6] https://en.wikipedia.org/wiki/Jass
[7] https://en.wikipedia.org/wiki/Truco
[8] https://en.wikipedia.org/wiki/Mate_(beverage)

Programmable Calculators

Adrian Kosmaczewski

2021-01-01

As far as I can remember, I have always been fascinated by computers. But in the crisis-ridden Argentina of the 1980s, buying one was beyond the reach of my mother. Actually, we did not even have a telephone back then.

The economy did not allow for much freedom of movement; the national currency was extremely devaluated, and was losing value every minute. Three different monetary units in a couple of years (the "Peso Ley 18'888[1]", the "Peso Argentino[2]", and the "Austral[3]") could not stop inflation, unbound and uncontrollable, from eating savings, sanity and happiness.

Thankfully, those were the best years of Argentine football. But that is another story.

My explorations within the realm of computers were limited to three simple sources of information: a couple of books, the Commodore 64[4] of one of my friends, and my beloved programmable calculators.

Books

My grandmother was a mathematician. Actually, she was one of the first female mathematicians ever to graduate in the mid 1920s from the University of Geneva. At home we had lots of science books; I have been exposed to technical literature since the very beginning. Most of it in German, French or English, neither of which I could read back then.

(Funny story, my grandmother was born merely three days after John Von Neumann[5], and both in the same Empire[6]; she was born in Philippopolis[7], Bulgaria, while he was born in Budapest, Hungary. But I digress.)

[1] https://en.wikipedia.org/wiki/Argentine_peso_ley
[2] https://en.wikipedia.org/wiki/Argentine_peso_(1983%E2%80%931985)
[3] https://en.wikipedia.org/wiki/Argentine_austral
[4] https://en.wikipedia.org/wiki/Commodore_64
[5] https://en.wikipedia.org/wiki/John_von_Neumann
[6] https://en.wikipedia.org/wiki/Austria-Hungary
[7] https://en.wikipedia.org/wiki/Plovdiv

One day, my mother, who was learning to use computers at her job, brought home my first computer book: a copy of the Spanish version of the "MS-DOS Users' Guide[8]" by Paul Hoffman and Tamara Nicoloff, published in 1984 by McGraw Hill.

(She was also learning Lotus 1-2-3[9] and dBase[10], but I do not remember her having anything else than a notepad with written notes about those.)

Actually, that MS-DOS guide was not even a real book, but a Xerox'd copy of the actual thing. This was the best my mum could afford. I still have it.

OK, that actually was not my first computer book. My mum had worked at IBM back in the 60s and 70s, and we had a sizeable amount of IBM Mainframe brochures at home. I wish I had kept them.

The MS-DOS book is actually still useful; well, maybe only if you use FreeDOS[11].

Commodore 64

In 1984, during my 5th grade of primary school, a friend in my class named Hernán Poletti got a Commodore 64 as a gift by her mother. It was, as far as I can tell, the only Commodore 64 in the neighbourhood, and needless to say, my friend became almost overnight the most popular person in the known universe.

Back then my only approach to programming consisted in reading code in programming magazines, of which a few were available in Argentina back then. My mother would find them in some newsstands in downtown Buenos Aires, to my greatest delight. But I had never seen code running; I had no idea how those lines of text could do anything cool on the screen of a computer monitor.

The arrival of that Commodore 64 changed it all; for hours in a row, we would type those lines of code on that clunky and clicky keyboard, and then we would save them in the cassette tape recorder. Yes, kids; back then, there were not even diskettes to save stuff into. We were using good old audio cassettes. As simple as that.

Later on, typing the `LOAD` and `RUN` commands on the screen would (sometimes) make the program actually do something on the screen. Most of the time it did not; and unfortunately, my debugging skills were not enough to solve the problems in those programs. I have never known if the bugs were due to my typing skills or to errors in the source code of the magazine; that point will remain in obscurity forever.

[8] https://www.worldcat.org/title/osbornemcgraw-hill-ms-dos-users-guide/oclc/11109098

[9] https://en.wikipedia.org/wiki/Lotus_1-2-3

[10] https://en.wikipedia.org/wiki/DBase

[11] https://www.freedos.org/

Casio fx-180P

All of this preparation was fruitful; my first exposure to programming happened around 1986, when I received a Casio fx-180P[12] calculator for my birthday, as a present from a distant European relative. I remember looking at this thing in dismay at first, with keys labeled with strange signs, like logarithms, sine, cosine, and other mathematical notations that I had never seen before.

Even worse, the manual was only in French and German. Let us just say that my discovery process of this calculator was slow and painful.

What puzzled me the most was that the keyboard had a button labelled `RUN` on the lower right side; and that was enough to lighten my curiosity. Could this be a small computer? I had seen that same button in the keyboards on computer shops, such as the Sinclair ZX Spectrum[13] or the Talent MSX[14].

Could this calculator be a computer, albeit a small one?

Much to my dismay, however, most of the functionality on this machine remained obscure. Until one day, during a rainy Saturday in Buenos Aires in 1989, I wrote my first program.

It was a simple program that allowed me to calculate the roots of the quadratic equation:

$$ax^2 + bx + c = 0$$

Which requires a well known formula:

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

The program, once run, would ask me for the three values a, b and c, and would proceed to give me the two possible roots, one after the other. The whole operation would mean just hitting the P1 key, which triggered the execution of the program; entering the three values, followed by the `RUN` button, and then the first value would appear. Pressing `RUN` a second time would show the second root of the equation.

Was it cheating? Of course it was. I hope Ms Elisa Quastler, my math teacher back in Argentina, will not be too angry to discover that this was the reason I could finish some tests in only a fraction of the time it took others.

[12] https://www.ithistory.org/db/hardware/casio-computer-co-ltd/casio-fx-180p
[13] https://en.wikipedia.org/wiki/ZX_Spectrum
[14] https://www.msx.org/wiki/Talent_TPC-310

First Program

Writing this small program was quite an exercise in itself, from a programming perspective at least. The Casio fx-180P, originally produced in 1980, had a (very) limited memory, and could only accomodate up to 38 instructions; each instruction being roughly equivalent to a single keystroke. It contained six addressable registers (known as K1 to K6), plus the standard "memory" accessed through the classic M+, M-, MR and MC buttons.

For reference, the instruction KIN reads a value into a register, while, predictably enough, KOUT does the opposite. DISP stops the execution of the program to show an intermediate value.

The program would simply simulate the sequence of steps that a human being would carry, storing the three coefficients in three of those addressable registers, then performing the operations in roughly this order:

```
KIN 1       # a --> stored in register 1
KIN 2       # b --> stored in register 2
KIN 3       # c --> stored in register 3
4
x
KOUT 1
x
KOUT 3
=
KIN 4       # 4ac --> stored in register 4
KOUT 2
x^2
-
KOUT 4
√
KIN 5       # sqrt(b*b - 4ac) --> stored in register 5
2
x
KOUT 1
=
KIN 6       # 2a --> stored in register 6
b
±           # Negative b
+
KOUT 5
=
/
KOUT 6
=
DISP        # First result displayed, hit `RUN` to continue
b
±           # Negative b
```

```
-
KOUT 5
=
/
KOUT 6
=
```

38 Steps

And here it is; probably written around June 1989, my first program, for the Casio fx-180P. It used all of the 38 steps available in the machine. It could not be longer than this, and maybe I could have "refactored" it, to make it shorter or faster, but for my particular needs, it was more than enough.

There was a big issue, though; if

$$b^2 < 4ac$$

then the program crashed; the Casio fx-180P did not have support for complex numbers.

HP 48GX

During my days as a Physics student in the University of Geneva I bought an HP 48GX[15] graphing calculator; compared to my previous Casio, this was a beast. And it is still quite an impressive piece of hardware, 27 years later.

For those wondering, yes, it is almost the same one used by Spiderman's father[16] in one of the movies; that one was an HP-48G, not the GX. The difference being that the GX had an extension bay for memory and program cards, as well as 128K of RAM instead of 32.

Oh, and did I mention it has an infrared port which can send and receive data using the Kermit protocol[17]? I still have the connection cable, with its RS-232 adaptor, allowing it to talk to any standard IBM PC of its era.

Le sigh.

Quadratic Functions in RPL

The same program as above, but this time in the RPL[18] language:

[15] https://en.wikipedia.org/wiki/HP_48_series
[16] https://www.techpoweredmath.com/spidermans-dad-hp-calculator-fan/
[17] https://en.wikipedia.org/wiki/Kermit_(protocol)
[18] https://en.wikipedia.org/wiki/RPL_(programming_language)

```
«
'C' STO 'B' STO 'A' STO
B 2 ^ 4 A C * * - √ 'E' STO
B NEG E + 2 A * /
B NEG E - 2 A * /
»
```

Although this might seem esoteric at first, it is very easy to understand once you know that RPL is a stack-based programming language (as is the whole calculator, by the way). In the code above, we just push values to the stack, and any binary operations will simply use the first and second values of the stack to perform its calculation.

The ST0 operation saves whatever value is in the second position of the stack to the variable whose name is in the first position.

As you might expect, the √ operation is the square root.

To execute this program, save it into a variable, push the values a, b and c (in this order) on the stack, and press the key that corresponds to the program. The values that appear after the execution are the much required solutions to the problem.

And of course, the biggest advantage of the HP 48GX compared to the Casio program above is that it has support for complex numbers, which is not the case for the previous program. This means that our little RPL program can solve all cases.

Coda

As I was finishing the preparation of this article, I found out there is a Texas Instrument calculator[19] that runs Python natively.

[19] https://web.archive.org/web/20210127004846/https://education.ti.com/en-gb/products/calculators/graphing-calculators/ti-84-plus-ce-t-python

Touch Typing

Adrian Kosmaczewski

2021-01-08

Probably the single most important skill that has always helped me when dealing with computers is the ability to type without looking at the keyboard.

I say it has been handy because, well, simply we spend a lot of time using keyboards, and they become the primary means of interaction with computers. Of course, these days we have lots of different input types; trackpads, mice, Wacom tablets, touchscreens… but the keyboard has kept a big deal of charm and usefulness, and touch typing has helped me master more about computers than I would have ever thought possible.

Advantages

I do not claim I am the best or faster typist around, mind you; I have seen much better and faster ones, and even for my own taste, I keep pushing the “backspace” key much too often.

But nevertheless, being proficient with a keyboard has provided several advantages.

Note Taking

The first useful thing about learning to touch type is simply being able to jolt down notes while listening to someone; and by listening I mean actually looking at the person in the eyes. Yes, just like a stenographer would do, listen and type.

Bragging

At some point, somebody will see you take notes without peeking at the keyboard and you are going to get some compliments for that. Be proud about it.

Coding

When you are coding, your brain must wire itself in order to solve the problem at hand. Most software writing is relatively simple; few of

us are actually sending probes to Mars or solving cancer or managing nuclear plants, so that our requirements for algorithms are quite simplistic.

But, nevertheless, coding requires attention, and if your brain spends time figuring out where the different letters are, you are going to pay less attention to the structure of your code, to your algorithms and other details. It is a simple logic; if the main thread in your brain has to pay attention to your hands, instead of just using a background thread to them, you are not going to do a great job. Keep your main thread free for analysis and coding.

Command Line

After 30 years of graphical user interfaces, I find fascinating that the most advanced visual technologies are still lagging behind the power of a good command line prompt. Call me old fashioned, but even if I enjoy the commodity of a good IDE from time to time, there is nothing like a good `Makefile` or a shell script to get things done.

In particular, I am fond of using tmux[1] for my command line needs. Productivity is key when the time is short, so I tend to choose technologies "command line friendly" technologies whenever possible.

Vim

Related to the previous point, there is my beloved text editor, Vim[2]. I wrote quite a few chapters about it in my book[3], so I do not need to say more right now; but suffice to say that touch typing has opened me the door of an absolutely delightful and incredibly powerful piece of software, probably the most stable and useful editor I have ever used in my life.

Suffice to say that Vim works best when your finger are glued to the keyboard. No mouse, nothing but keystrokes and power.

Keyboard Fetish

Oh, speaking about keyboards and power, why deny it; you will develop a keyboard fetish. It is part of the game.

You are going to realize that some keyboards are better than other, in subtle ways; some of them just require too much effort to write even the simplest of sentence. Some others are simply too soft, and will make your writing even worse, with repeated letters all over the place. Then there is the height, the width, the color, maybe, and finally, even the smell. You are going to buy them, collect them, talk about them,

[1] http://tmux.sourceforge.net

[2] http://www.vim.org

[3] /books/Tales_Of_Editors_And_Keyboards/Tales_Of_Editors_And_Keyboards.html

discuss their relative merits, people are going to hate or admire you for them, and this is just part of the game.

My personal favorites at the time of this writing are: the Das Keyboard Ultimate 4, the Apple Wireless keyboard, and in general, most Microsoft and Logitech keyboards. Historically speaking, I used to love the classic IBM PC keyboards, and I would love to own a Model M one day. I can say the same about the Apple Extended Keyboard II[4]. I would love to own those.

Ergonomic Keyboards

I used to own a Microsoft Ergonomic keyboard back in the nineties; I loved writing with my wrists in a natural position. I find it sad that there are no good ergonomic keyboards around these days anymore. I think that the idea of an ergonomic keyboard was absolutely fantastic, and having used them, I could really see myself writing this book in one of them.

How To Learn

I learnt touch typing at school, in Argentina, in 1989. At home we had a beautiful Remington Typewriter, one that belonged to my grandmother, and I loved to write in it. However, my mother, who was a secretary, would type in her IBM Selectronic typewriters at work at incredible speed, and I wanted to know how to do that.

In 1989, when I was in 3rd year of high school, we had the option to choose between a course in drawing and another about touch typing. I chose the latter. In our school there were no computers (remember, this is the Argentina of the '80s I mentioned so many times in this blog) so we had a large room stocked with old mechanical typewriters from various brands and origins, in the last floor of the school building.

We would go there every Friday morning, during one hour, for a whole year, and the machines had metal plates on top of the keyboards, so that we would not see our hands while writing. The exercises consisted of filling sheets of paper with sequences like `abababab cdcdcdcd efefefefef` and more, so that our fingers would remember the position of the different keys once and for all.

Keyboard Layouts

My personal experience shows that the American keyboard layout (commonly known as "QWERTY") is the one and only one that helps me write good code. The best programmers I know use it, the best software I wrote was with it. Thankfully, the typewriters we used at school had either the American layout, or the Latin American one (which includes the quintessential "ñ" letter.)

[4] http://www.wired.com/2008/05/a-tale-of-two-k/

The reason for this choice of the American layout is that many of the most popular programming languages (among which one can easily spot Java, C, C++, Objective-C, JavaScript and others) all use some combination of symbols (such as curly and square brackets, semi-colons and apostrophes) which are immediately available in the American layout.

In stark contrast, I argue that the French ("AZERTY") and the Swiss keyboard layout ("QWERTZ") are the least appropriate for writing code. I have seen students in my training classes struggle with laptops with those keyboards. Hitting "ALT GR+6" and "ALT GR+7" to get a set of curly brackets seems to me a useless waste of energy, and if you have to repeat this every 10 seconds, you are going to get very angry, very soon.

And you do not want to code angry.

Keyboard Suggestions

Here goes a list of interesting keyboards out of the ordinary; find your own favorite here:

- Ergodox EZ[5]
- Moonlander Mark I[6]
- Keyboardio Atreus[7]
- Razer BlackWidow V3 Pro[8]
- Das Keyboard[9]
- Ultimate Hacking Keyboard[10]
- MAX Keyboards[11]
- Pimoroni Keybow[12]
- Keychron[13]
- WASD Keyboards[14]
- Happy Hacking Keyboard[15]
- Unicomp PC "Model M"-like Keyboard[16]
- GH60 Programmable Keyboard[17]
- KBDfans[18]

[5] https://www.ergodox.io/
[6] https://ergodox-ez.com/
[7] https://shop.keyboard.io/
[8] https://www.razer.com/gaming-keyboards
[9] https://www.daskeyboard.com/
[10] https://ultimatehackingkeyboard.com/
[11] http://www.maxkeyboard.com/
[12] https://shop.pimoroni.com/products/keybow?variant=21246333190227
[13] https://www.keychron.com/
[14] http://wasdkeyboards.com
[15] https://www.hhkeyboard.com/
[16] https://www.pckeyboard.com/
[17] http://blog.komar.be/projects/gh60-programmable-keyboard/
[18] https://kbdfans.com/

- CODE Mechanical Keyboards[19]
- Matias Ergo Pro[20] and Tactile Pro[21] (the latter for Mac users, particularly those feeling nostalgia about the Apple Extended[22] keyboard)

For the record, I used my Das Keyboard 4 Ultimate[23] to write these lines. Highly recommended.

PS: If you liked this article, you will enjoy my book “Tales of Editors & Keyboards”, freely available[24] in this website.

Update, 2022-11-04: Just found out about the Steelseries gaming keyboards[25].

[19] http://codekeyboards.com/
[20] https://matias.ca/ergopro/programmable/
[21] https://www.matias.ca/tactilepro/
[22] https://en.wikipedia.org/wiki/Apple_Extended_Keyboard
[23] https://www.daskeyboard.com/daskeyboard-4-ultimate/
[24] /books/Tales_Of_Editors_And_Keyboards/Tales_Of_Editors_And_Keyboards.html
[25] https://steelseries.com/gaming-keyboards

Bugs

Adrian Kosmaczewski

2021-01-15

Somewhere between the end of the Second World War and the beginning of the Cold War, when the enemy started speaking Russian instead of German, a US Navy programmer was working in an early computer, trying feverishly to solve a problem in a program.

Finally, says the legend, Mrs. Grace Hopper opened the computer chassis and discovered, between the cables and the bolts, a dead cockroach.

A bug, found during the first debugging session ever in the history of mankind.

This anecdote is illustrative in many ways, mainly because it contains almost no mention of the program that did not work – of course, its contents might have been considered state secret or something like that, but the truth is that the "bug" was actually a foreign body, completely unrelated to the program being executed.

Software is mostly bugs

In your professional life, however, most of the code you will write and use is buggy. Most of the libraries that you will download and use in your web pages contain bugs of very different kinds. Actually, working software will be the exception to the rule; most of the software does not work, and will never truly work all the time.

Why this is so you ask? Well, to begin with, a program is a promise that you cannot keep. You describe a world, a sequence of steps, a structure in memory that must be operated upon following a certain way and in a certain order. But the world is imperfect; maybe the computer running your software 10 years from now will have new security mechanisms that will block the system calls you are using; maybe it will use a certain type of memory chips that do not allow certain kinds of allocations; maybe there will not be enough memory at all.

But you, in your code, you are programming your application against an imaginary piece of hardware. You are going to be making tons and tons of assumptions, and you know what? That is ok. Everybody does

that. The difference between a junior and a senior developer is the decreasing amount of assumptions made in one's head.

No guarantees

So, the thing is, no matter which programming language you choose, there are no guarantees.

Yet, in spite of all of the evidence, many developers feel a warm fuzzy feeling when using statically typed languages. It is like if they felt more protected; you know, this variable is always going to be a pointer to a string, or this other variable is always going to be an integer. They put up with ridiculously long compilation times, complex syntaxes, casting objects up and down their class hierarchies in order to do things.

Compile time vs. Run time

While other developers, maybe more aware of the reality of the world, happily write and publish applications in languages such as Ruby, Python, JavaScript or Objective-C, knowing well that if something can go wrong, it will go wrong.

C++, Java, Swift and other languages scoff at the thought of developers keeping track of the objects at the end of the variables, and make sure that every method call, that every function and every parameter and every possible combination of templates and generics actually makes sense to an increasingly complex theorem, verified at every compilation.

Yet, in all of its glory, even these allmighty languages have to declare themselves lost at some point, and they implement vtable lookups in their objects anyway, because there is always some situation in which the resolution of the polymorphic method to be called cannot be determined at compile time, and we have to cross fingers and pray that everything will be OK at runtime.

An analogy

When I used to teach iOS programming to developers, I used a very simple analogy that always made me laugh.

Somehow I thought of C++ as an east-coast kind of language: uptight, control-freak, obsessed with detail and verification, and without any trust whatsoever. If C++ was a person, it would be a rich financial trader in Manhattan, with a nice suit and a nice car, and a perfectly controlled life around him.

A bit like American Psycho, if you see what I mean.

On the other hand, I used to describe Objective-C as a pure west-coast kind of language, a product of the 70's, a relaxed language that

would make almost no verification whatsoever of your types, while at the same time it would be smoking pot and listening to Led Zeppelin out loud. Objective-C would be a 1971 hippie in the middle of San Francisco, protesting against the war in Vietnam and enjoying life as it comes.

A bit like American Pie, if you see what I mean.

Learning

What I recommend to you, faced with strongly and weakly, statically and dynamically typed languages, is that you start your career with a really strongly typed one. It will make your beginnings easier, as long as you can understand the error messages of your compiler – and believe me, they can be quite hard to understand sometimes.

But as you grow up, let's say 10, 15 years from now, move to more dynamic languages. Statically typed languages, maybe something like Go (not particularly C++ to be honest, buy why not) that will provide you with a simpler mental model to follow at the beginning. The compiler will basically take you by the hand and guide you until you have a working piece of software.

However, let your inner instinct guide you towards more relaxed languages as you move forward. Open up your mind, and enjoy the tremendous freedom offered by a language whose compiler does not stand up in your way.

Job Interviews

Adrian Kosmaczewski

2021-01-22

If you work in this industry for a certain amount of time you are certainly bound to suffer, at some point or another, the delicious experience of the programmer job interview.

Countless books have been written about the subject; everybody has their own opinion about it. And of course, dear reader, you are now going to hear my own: they all suck.

The programmer job interview is, at best, an exercise in futility. People who have never met before must decide, in a series of one to ten meetings of one hour each, if their future belongs together or not. Think speed dating minus the sex, minus the good food, minus the fun, minus everything, plus a contract and a salary.

Of course this process, largely driven by heuristics of different kinds, the least of which is rationality, can sometimes yield positive outcomes; yes, you can land a nice job through an interview. Inversely, if you are the employer, it might happen that you are lucky and you find the right employee.

But make no mistake, there is no scientific method in the job interview. There is guts, no data, no rationality. The problem with the job interview lies, once again, in hypocrisy. The moment things go south is when companies think they have a well rounded interview "process."

How to prepare

The short answer is: do not. Just be yourself. The best job interviews are actually conversations between two like-minded people. You want smart people in front of you, so just be yourself. I think that you do not prepare for a job interview; you just become, every day, a better software developer. You keep reading, you keep learning, and then every so often you go to an interview.

If you are yourself in an interview, you are making everyone a favour. They will see, in those short interactions, a faithful sample of your personality; your flaws and your capabilities. Being yourself sends a clear message: you do not settle for mediocrity or hypocrisy. You

want like-behaved people in front of you; you know what you want, and you are actively looking for it.

Of course, it is to expect that the interviewers are also being themselves, but that is beyond your reach, so remember that the only thing you can do is pay attention, and, in the worst of cases, politely dismiss the interview and walk out.

It is a two-way street

Most developers, particularly young ones, think that they should bend their wishes to any extent in order to get that job. Hint: you do not need to do that. A job interview is also a way for you to find out whether you want to work at that place, too, so you should be alert and watch out for any red lights.

Red lights

Talking about signs that you should not sign up at a company, here is a couple few:

The interview is all about the company, little about you

Some employers are so full of themselves (this is typically true with large, well known companies in Europe) that they will spend the whole time talking about their high standards, their achievements, and the kind of people that works for them. Pay attention to the wording: you can detect these folks because they say “people work for us” instead of “with” us. This subtle but important word choice reveals a lot about a company.

The interviewer only pays attention to gaps in your CV

For some reason, some HR managers think that any period of time you do not mention in your CV is worth investigating. Maybe you took one year off, maybe you had family problems, maybe you just wanted to not work for a while. It is none of their business, and I actually think it is rather impolite (and, dare I say, the practice should be banned and outlawed.) Unfortunately in Europe personal questions are not illegal, so brace for impact. The interview should, at any time, be about what you have done, not what you have not done.

Few of the people interviewing you are technical folks

It is OK to have the occasional HR manager assessing your personality, of course; but more than 75% of all the people you are going to see in an interview should be developers, and not any developers, but potential co-workers from the team you would be assigned

to. An interview is a rather short process, so the company should be optimizing the time spent in them.

Watch out for simple lacks of decency

Managers arriving late to the meeting room where the interview takes place; staff who do not even offer you a glass of water when you arrive to their offices; wrong directions to the location of the interview; comments about your appearance, nationality, or accent; managers ranting about their current staff or employer; inappropriate comments about race, gender, nationality or other non-technical factors.

As a rule of thumb, at the first occurrence of such events, one must wrap up the interview, greet the team and get out. If they behave this way in our first meeting, imagine what it will be when you sign the contract.

Stupid technical questions

Frankly, it is 2015 and companies are still asking how to write a linked list using C, on a whiteboard, without documentation, in one minute. This is nonsense. Whenever you are asked a question that could be answered by opening a book or searching in Stack Overflow, say so politely.

I tend to favor conversational topics, which I think make for a much better interview: design patterns, architecture, code quality issues, experiences in such and such frameworks, and so on. Conversational questions allow both parties to enter, guess what, in a conversation. And through that conversation, both parties can discover the other, the person on the other side of the table.

Most HR managers think that their job consists of finding people that allows them to cross checkboxes. That is wrong. In the words of Joel Spolsky, they should be looking for smart people who gets things done. And a sure way to find them is to actually talk to them, like human beings. Algorithms and data structures? Sure, there are plenty of literature about those; let's move on.

Server-Side JavaScript in 1997

Adrian Kosmaczewski

2021-01-29

Back in 1997 I was earning some cash writing Active Server Pages[1] in that mutant programming language called VBScript[2]. On the other hand, Microsoft had reverse-engineered the JavaScript compiler inside Netscape Navigator (there was no spec, after all), and they created a dialect of it called JScript[3] (they could not name it JavaScript because lawyers) and made it compatible with their COM+[4] runtime model.

These were the 90s; the future was COM+ components using `IUnknown`[5] and `IDispatch`[6] interfaces, so that you could create applications using COM components created in C++, stitched together with a scripting language, like JScript of VBScript. A bit like how Objective-C wraps low-level C libraries, a bit like how Swift wraps low-level Objective-C frameworks. Always the same idea.

Of course the future turned out to be .NET, not COM+, and now I think their next platform should be called ORG just for the sake of playing with domain names.

I know, weird times and even weirder names. But the truth is that in 1998 I was literally writing server-side JavaScript, but as you can imagine it was neither convenient nor performant nor hype. Actually people hated JavaScript vehemently.

It is not like there were many choices. There was Cold Fusion[7], but programming with HTML tags still rings creepy to me. PHP was not yet an option. WebObjects[8] might have been one, albeit a much more expensive one. Like, 50'000 USD per server license. And we did not know anything about Objective-C. Lisp[9] might have been another, but we knew even less about it.

So VBScript it was.

[1] https://en.wikipedia.org/wiki/Active_Server_Pages
[2] https://en.wikipedia.org/wiki/VBScript
[3] https://en.wikipedia.org/wiki/JScript
[4] https://en.wikipedia.org/wiki/Component_Object_Model
[5] https://en.wikipedia.org/wiki/IUnknown
[6] https://en.wikipedia.org/wiki/IDispatch
[7] https://en.wikipedia.org/wiki/ColdFusion_Markup_Language
[8] https://en.wikipedia.org/wiki/WebObjects
[9] http://www.paulgraham.com/avg.html

Why did people hate JavaScript so much? That is a question that I have asked myself quite a few times during the past 22 years. I think it has to do with broken promises. JavaScript looks like Java, it even has a name that sounds like Java, it has curly brackets and semicolons like Java, yet the `this` pointer usually points somewhere else than where one expects. Hence people gets pissed off, and they rejected all contact with the language for as long as they could.

However, life is a bitch, and JavaScript being the only programming language to write stuff running on a web browser, that meant sooner or later you would have to deal with it, anyway, no matter how much you hated it. We had to wait until 2001 for Douglas Crockford to be the first person to actually understand JavaScript[10] and then expose its good parts seven years later[11] in a now absolutely classic book.

So people came up with many strategies to cope with the pain. The earliest attempts around 2010 were CoffeeScript[12] (targeting Python lovers) and Cappuccino[13] (for the Objective-C demographic). Microsoft's (once again) TypeScript[14] is one of the latest and, at the time of this writing, one of the most popular options these days, together with Elm[15], Kotlin[16], ClojureScript[17], LiveScript[18], Amber[19] (for Smalltalk folks), Dart[20], Fable[21] (for F# aficionados), Opal[22] (for Ruby fanatics), and there are new ones popping every week.

Clearly nobody wants to deal with plain JavaScript. I understand them. But I digress.

On the server side, however, ASP had a surprisingly simple programming model. Everything was quite procedural, one mixed and matched HTML and code as you would in PHP nowadays, and you could use five big global objects, named after their respective classes, to make stuff happen: `Application`, `Request`, `Response`, `Server`, and `Session`. You could put global state at application or session level, the runtime provided basic I/O functionality, and lots of functions to perform whatever task you wanted to do.

And if you needed something fancier, you could always buy a COM+ component from a third party, install it in your server and instantiate it to perform some tasks, like sending e-mail from the server[23], which

[10] https://www.crockford.com/javascript/javascript.html
[11] https://www.oreilly.com/library/view/javascript-the-good/9780596517748/
[12] https://coffeescript.org/
[13] https://www.cappuccino.dev/
[14] https://www.typescriptlang.org/
[15] https://elm-lang.org/
[16] https://kotlinlang.org/
[17] https://clojurescript.org/
[18] https://livescript.net/
[19] https://www.amber-lang.net/
[20] https://dart.dev/
[21] https://fable.io/
[22] https://opalrb.com/
[23] http://aspemail.com/manual_02.html

was not available off-the-box in the ASP runtime. You read right.

VBScript did not allow you to create classes, at least not in the sense that you might expect; there was a `Class` keyword, yes, but it merely created in-memory structures, which for reasons nobody understood could not be stored neither on a Session nor on an Application object, and they were as such only useful as parameter or return types for functions. Quite useless as a matter of fact, mais c'est la vie.

To be fair, there was a COM object implementing a simple hashtable object one could use from within VBScript, giving you something like this[24]:

```
Function DicDemo
   Dim a, d, i, s   ' Create some variables.
   Set d = CreateObject("Scripting.Dictionary")
   d.Add "a", "Athens"   ' Add some keys and items.
   d.Add "b", "Belgrade"
   d.Add "c", "Cairo"
   a = d.Items   ' Get the items.
   For i = 0 To d.Count -1 ' Iterate the array.
      s = s & a(i) & "<BR>" ' Create return string.
   Next
   DicDemo = s
End Function
```

As you might expect, the same objects were available from JavaScript; the difference was that instead of using `CreateObject()` you would use `new ActiveXObject()` instead, passing the name of the COM class as a parameter. All very dynamic.

```
function ItemsDemo()
{
   var a, d, i, s;                  // Create some variables.
   d = new ActiveXObject("Scripting.Dictionary");
   d.Add ("a", "Athens");              // Add some keys and items.
   d.Add ("b", "Belgrade");
   d.Add ("c", "Cairo");
   a = (new VBArray(d.Items())).toArray();   // Get the items.
   s = "";
   for (i in a)                   // Iterate the dictionary.
   {
      s += a[i] + "<br>";
   }
   return(s);                     // Return the results.
}
```

I hope you have not missed that `new VBArray()` thing in line 8.

The truth is that even if we could have used JavaScript both on the client and the server side, we wrote most of the backend in VBScript.

[24] https://docs.microsoft.com/en-us/previous-versions/windows/internet-explorer/ie-developer/windows-scripting/8aet97f2(v=vs.84)?redirectedfrom=MSDN

The thing is, many features in the ASP runtime did not work correctly on JScript, like for example disconnected `ADODB.RecordSet` objects. Well, theoretically you could make them work, but the page that explained how no longer exists[25], and it only worked in later versions of IIS and ASP.

Also, we were early adopters of the "Internet-Based Programming" methodology, way before Stack Overflow existed, waaaaay before Joel Spolsky started blogging[26]. Since most of the snippets you found on the web were in VBScript and we were really lazy developers, well, the backend ended up in VBScript and that was it.

Somebody even came up with a tool for unit tests in VBScript[27] in 2005, but by that time I was already doing something else.

[25] https://support.microsoft.com/en-us/help/289531/how-to-create-ado-disconnected-recordsets-in-asp-using-vbscript-and-js

[26] https://www.joelonsoftware.com/2000/04/06/things-you-should-never-do-part-i/

[27] https://web.archive.org/web/20160527073004/http://xt1.org/scriptunit/

AJAX Before Time

Adrian Kosmaczewski

2021-02-05

In 2002 I moved back to Switzerland, and found a first job as a developer in a company making a super expensive product nobody needed. It tanked one year later for well known reasons, but in the meantime I got to see the most complex codebase I had ever met, in this case in JScript.

It was humongous; we are talking here about an ASP application in the hundreds of thousand lines of JScript code, in impeccable object-oriented style, with all the Gang of Four design patterns showcased in an extreme example of good architecture. Do not get me wrong; for I am not being ironic here. It was a wonderful team with kind and brilliant people, and I learnt a lot during my year with them. The web application, however, took a sensible time to startup and to run; it had serious performance issues. Its complexity had a huge cost.

But the most fascinating thing about this codebase what that it had a hidden secret deep inside. See, the app screens would update themselves without reloading the page. This was 2002. I had never seen anything like this before. The first time I noticed it I thought my eyes had skipped a frame or two, but then I realized that every so often the UI was changing; things were appearing and disappearing, text and images would change dynamically. I first thought they were JavaScript animations – like the ones you could create with Macromedia Dreamweaver[1] back then, with long chunks of JavaScript separated with rather horrendous `if (IE) {` blocks.

But no, this something else. This was actual data being reloaded and reshuffled and the whole thing was alive. Then I thought that they might be using a hidden `<IFRAME>` to get the trick; so I grabbed my beloved EditPlus[2] and I started digging in the codebase.

What I found was astonishing. There was a bit of code at the far end of the call stack, with the following line:

```
var request = new ActiveXObject("Microsoft.XMLHTTP");
```

That little object had event handlers, and those handlers were updating the user interface accordingly, every time that the backend had

[1] https://en.wikipedia.org/wiki/Adobe_Dreamweaver
[2] https://www.editplus.com/

changed its state for whatever reason.

I saw AJAX in action for the first time in 2002, three years before Jesse James Garrett[3] coined the name in an article in his blog[4].

By the time Mr. Garrett wrote his article, Safari, Firefox and Opera had also added a component with pretty much the same interface, which allowed developers to write code that looked like this[5]:

```
if (window.XMLHttpRequest) {
    //Firefox, Opera, IE7, and other browsers will use the native object
    var request = new XMLHttpRequest();
} else {
    //IE 5 and 6 will use the ActiveX control
    var request = new ActiveXObject("Microsoft.XMLHTTP");
}
```

A few years later, Prototype.js and jQuery were born to hide this `if/else` statement inside of a library, among other functionalities. The rest is history, and it was called “Web 2.0”, whatever that meant.

[3] https://en.wikipedia.org/wiki/Jesse_James_Garrett

[4] https://web.archive.org/web/20160120235937/http://adaptivepath.org/ideas/ajax-new-approach-web-applications/

[5] https://developer.mozilla.org/en-US/docs/Web/API/XMLHttpRequest/Using_XMLHttpRequest_in_IE6

Visual J++

Adrian Kosmaczewski

2021-02-12

Once upon a time, there was a programming environment made by Microsoft called Visual J++[1]. It was their attempt to do with Java what they had done with JScript before, and to be honest, it was quite cool. You could compile and run Java code on Windows with a very good IDE - this was 5 years before IntelliJ released IDEA[2]! It generated much faster binaries than what the official Java compiler from Sun produced. Developers could access functionality inside of packages starting with the `microsoft.` name, but that of course that kind of broke the whole point of Java which is to make cross-platform stuff that you only write once and then you run everywhere.

So at some point Sun got fed up and sued Microsoft, who promptly threw Visual J++ away and came up with something more interesting, which was C# and .NET, which was essentially like Java but with one major difference: it could only run in Windows. Which in 2000 was still an understandable proposition.

Both C# and Java have very similar "Hello, World!" implementations:

```java
public static void main(String[] args) {
    System.out.println("Hello, World!");
}
```

```csharp
public static void Main(string[] args)
{
    Console.WriteLine("Hello, World!");
}
```

And we all know that "Hello, World!" is the golden standard for programming language comparison. Show me your "Hello, World!" and I will extrapolate enough facts for a whole subreddit.

From a historical point of view, it is quite obvious that the major difference between them stem from just one person: Anders Hejlsberg, who was also the creator of Turbo Pascal, Delphi, Visual J++, and would later bring TypeScript to the world. Talk about a resumé. And since in Pascal one uses `PascalCased` entities pervasively, not only for classes but also for methods, so C# came to be.

[1] https://en.wikipedia.org/wiki/Visual_J%2B%2B

[2] https://www.jetbrains.com/lp/intellijidea-20-anniversary/

Well, there is also the position of the opening curly bracket. That is the second major difference.

The dichotomy between Java and .NET was so strong, that even a popular website such as TheServerSide.com[3] begat TheServerSide.net[4] as a way to control the clash of matter and antimatter.

After the release of .NET, Microsoft came up with a successor of Visual J++, called J#[5] which as the name suggests was a Java compiler for the .NET runtime – but it was discontinued a few years later.

See, the only thing that everyone wanted Microsoft to do with .NET in 2002 was to make Visual Basic 6 compatible with Visual Basic.NET. And that is precisely the one feature that enterprise developers never got. Even worse, both languages were largely incompatible with each other, to the extent that even copy-pasting code from one to the other did not work. Microsoft was very vocal in that you could call COM components from within both languages, but the truth is that nobody made COM components anyway; most apps were written in VBScript or Visual Basic 6, and neither of these had a decent migration path towards .NET. That is what I call spitting on the face of your biggest customers.

Or, as Ballmer would say, developers, developers, developers.

So the beginning of the 21st century was marked by yet another Great Rewriting™®©. It happens every so often. Well, not that often for COBOL or Fortran, but still.

But to be fair, starting with the second version of C# in 2005, Java and C# started to diverge a lot. Which is an euphemism to say that Java stagnated and C# evolved. So much that many Java developers started to use anything but Java (the language) in their Java (the runtime) applications; Clojure[6] for functional programming purists, Scala[7] for backend services, Kotlin[8] for Android applications, JRuby[9] for scripting and Groovy[10] for build systems. Heck, at some point you could even run PHP[11] inside the JVM if you really wanted to.

Of course there is a team of developers working overtime in the basement of a bank in Zurich, still maintaining the Java applets forming the core of an intranet only compatible with Internet Explorer 6 that, for some weird reason, refuses to die in dignity. If you are one of those developers, reading this note during your coffee break, please shout. We will come to rescue you.

[3] http://www.theserverside.com/
[4] http://www.theserverside.net/
[5] https://en.wikipedia.org/wiki/J_Sharp
[6] https://clojure.org/
[7] http://www.scala-lang.org/
[8] https://kotlinlang.org/
[9] http://jruby.org/
[10] http://groovy-lang.org/
[11] http://caucho.com/

Java got lots of backlash in the past 15 years, starting with Steve Yegge[12], who has been very vocal against it[13], and now apparently loves Kotlin[14]. But just like in fashion, Java is kind of cool again lately, in particular thanks to Quarkus[15], which is basically a Kubernetes-ready Java runtime.

And by putting the word “Java” next to “Kubernetes”, Red Hat won the buzzword war. Which is precisely what IBM expected for the 34 billion USD price tag.

[12] http://steve-yegge.blogspot.ch/2006/03/execution-in-kingdom-of-nouns.html

[13] http://steve-yegge.blogspot.ch/2007/12/codes-worst-enemy.html

[14] http://steve-yegge.blogspot.ch/2017/05/why-kotlin-is-better-than-whatever-dumb.html

[15] https://quarkus.io/

Thirty Years

Adrian Kosmaczewski

2021-02-19

As the taxi rushed away from my old home along Avenida del Libertador[1], I looked through the rear windshield for one last time. My mother barely acknowledged my gesture. She was silent, and most probably did not want to turn around. She hid her eyes behind her sunglasses, trying not to think about what laid ahead of us, in a journey that, a few days later, would take us to Europe.

It was February 1991. The weather was warm, not hot, and very sunny. The USA were starting a campaign[2] to kick the forces of Irak out of Koweit. Argentina was in a deep economic recession. Again[3].

As I recall these memories from my childhood, seeing my first home fade away in the distance, I realize that Vicente López[4] was not yet the chaotic, impersonal mess of highrise buildings that it is today. There were still a few safeguards in place, and although there were quite a few very high constructions on the west side of the avenue, they all enjoyed a fantastic view of the Rio de la Plata[5].

As did my mother's apartment; a nice one-bedroom flat she had bought in 1968, thereby meeting my father. She was a secretary working in IBM Argentina[6]. He was a young student in the Faculty of Architecture of the Universidad de Buenos Aires[7], making ends meet by selling real estate in the northern neighborhoods of the Greater Buenos Aires Area[8].

As fate had it, my mother bought appartment "B" in the seventh floor of the building in Avenida del Libertador 1574 (between Gavito Automotores and the YPF gas station) somewhere in 1968 (that building is clearly visible in the middle of this picture[9] on Wikipedia). She mar-

[1] https://en.wikipedia.org/wiki/Avenida_del_Libertador

[2] https://en.wikipedia.org/wiki/Gulf_War

[3] https://en.wikipedia.org/wiki/Economic_history_of_Argentina

[4] https://en.wikipedia.org/wiki/Vicente_L%C3%B3pez%2C_Buenos_Aires

[5] https://en.wikipedia.org/wiki/R%C3%ADo_de_la_Plata

[6] https://www.ibm.com/planetwide/ar/

[7] https://en.wikipedia.org/wiki/University_of_Buenos_Aires

[8] https://en.wikipedia.org/wiki/Greater_Buenos_Aires

[9] https://web.archive.org/web/20131225140335/http://upload.wikimedia.org/wikipedia/commons/7/7c/Aerial_view_of_Buenos_Aires_and_Rio_de_la_Plata%2C_2009-03-18.jpg

ried my dad in 1971. I was born in 1973. They divorced in 1974.

I digress.

Vicente López

Vicente López[10] was a very nice place to live. In summer one could hear the cicadas on top of the Jacarandas. Lots of bitter orange trees would furnish the neighborhood with the smell of fresh marmalade. I rode my bike across the streets, avoiding the few bus lines that crossed it (most notably lines 161 and 21), and enjoying the sights. Avoiding the river[11], too, because it was too polluted to be approachable at less than 100 meters. Some days the stench was unbearable.

Ours was an area of famed pizzerias, restaurants and cafés. There was “Kaskot’e”, in the corner of Vernet and Libertador, really close from the Centro Asturiano sports club. One of my preferred restaurants was “Carlitos”, in the corner of Vergara and Libertador, whose owner and cook was exactly the same Carlos Ciuffardi[12] from Villa Gesell[13]. Once a month my mum took me to the Pumper Nic[14] in the corner of Yrigoyen and Libertador to have a “Mobur” (a sandwich with ham and a fried egg) with “Frenys” (French fries). I even celebrated my 8th birthday with my schoolmates in that Pumper Nic. And for a relaxing coffee with friends, I can recommend the famous Café de Paris[15] which is still in the corner of Ramón Melgar and Azcuénaga, and is still as charming as it was back then.

It was also a neighborhood of famous sport clubs: the Centro Asturiano (of which my mother and I were members); the Círculo Trovador; the Centro Lucense; the Club Banco Provincia; those were all near my home. A bit further away, along the Zufriategui street, was the Banco Nación Club, with its famed Rugby team. And there were quite a few military clubs along the river as well.

To be honest, as much as I liked the place, I walked those streets with eyes in my back, avoiding to be mugged, or worse. By the end of the 1980s it was no longer safe to live there. Actually, it felt like none of Argentina was safe anymore.

Decisions, Decisions

In August 1990 my mother and I took the decision of leaving the country to go to Switzerland. Well, the decision was years in the making,

[10] https://en.wikipedia.org/wiki/Vicente_L%C3%B3pez%2C_Buenos_Aires

[11] https://en.wikipedia.org/wiki/R%C3%ADo_de_la_Plata

[12] https://www.welcomeargentina.com/villagesell/carlitos-rey-del-panqueque.html

[13] https://en.wikipedia.org/wiki/Villa_Gesell

[14] https://en.wikipedia.org/wiki/Pumper_Nic

[15] https://www.tripadvisor.com.ar/Restaurant_Review-g1438379-d12355828-Reviews-Cafe_de_Paris-Vicente_Lopez_Capital_Federal_District.html

really; after all, we both had the Swiss passport. She had got it from her father, and I had got it from her.

With that objective in mind I started learning French again in 1988. I say again because when I was a kid my grandmother used to speak to me in French. And then I started primary school at a very expensive multilingual school[16] in the neighborhood of Martínez[17]. I only stayed there for three years, from 1978 to 1980. It was prohibitively expensive.

So I went to public school for the rest of the 1980s. Starting in 1981, I went to the "Escuela número 8" in the corner of Maipú and San Martín, and in 1987 I started high school in the "Colegio Nacional de Vicente Lopez Juan Pablo Duarte y Diez", in Agustín Alvarez street.

Going back to study French was a way to recover those memories from when I was a kid. My grandmother had passed away in 1985 after a few years of chronic illnesses and health troubles. I had not spoken French since the early 1980s, and I had forgotten most of it.

Funny anecdote: my French teacher was Eleonora, a lifetime friend of my mother, and incidentally the mother of Beta Suárez[18], a well known influencer in Argentina nowadays. As a matter of fact, my grandmother helped Eleonora pass the exam in the Alliance Française to become a teacher, and 20 years later, Eleonora taught me French.

Tipping Point

In 1989 a new government came into power in Argentina. My mother was working in a public company[19] of the energy sector, one of the myriad of enterprises to be "privatised" in name of neoliberalism, efficiency, and corruption. She was going to lose her job. She knew it.

She sold her appartment, today worth at least 90'000 USD, for what was the market price at that time, a mere 15'000 USD (around 30'000 USD in 2021, adjusted by inflation). She actually got 5'000 more for it because she had subscribed to the "Megatel Project"[20], which had provided a phone to our home. Another anecdote: in 1989, a phone rang in our appartment for the first time, ever. An orange phone, with a dial, as analog as it gets. Having a phone was so rare, it could increase the value of your home.

That means, yes, she had lived for almost 20 years without a telephone at home. Can you believe it?

We went to the Swiss embassy[21] in downtown Buenos Aires, in the

[16] https://www.cfam.edu.ar/
[17] https://en.wikipedia.org/wiki/Mart%C3%ADnez,_Buenos_Aires
[18] https://www.betasuarez.com/
[19] https://es.wikipedia.org/wiki/Hidronor
[20] https://en.wikipedia.org/wiki/ENTel
[21] https://www.eda.admin.ch/buenosaires

Avenida Santa Fé, and asked for information. They were polite but did not provide much data. A few tourist guides about Geneva, a list of hotels, but not much more.

We were on our own.

Takeoff and Landing

On Tuesday, February 19th, 1991, around 9 AM local time, our KLM flight took off from EZE[22]. The next day, after a few hours layoff in AMS[23], we landed in GVA[24].

It was Wednesday, February 20th, around 3 PM local time.

That had been my first transatlantic flight ever.

Nobody was waiting for us in the airport. Nobody from my mother's family knew we were there. Actually, to the exception of one cousin, nobody even dared have a coffee with her. I do not know what happened in her family, but for some reason they hated her brother intensely, and by transitive property, her as well, and I suppose me too.

Upon arrival, we left our bags in lockers at the Geneva Airport, and took a train to downtown Geneva. The only thing we knew is that there was a tourist office in Cornavin[25], Geneva's main train station. The lady in the counter booked a hotel for us, we took a cab, and dropped our handbags at what would be our new home for the forseeable future.

That hotel no longer exists, and thankfully so. It was called Hotel Adris, in the Rue Abraham Gevray, not far from the lake, in the neighborhood of Pâquis[26]. Quite crappy but cheap. No need to say more.

We went back to the airport, grabbed our bags, got back to the hotel, and then left for a small walk, to get to know the neighborhood a bit. We ate a pizza in a small restaurant in the Rue des Pâquis, between the Rue de Monthoux and the Rue de Zurich. That was our first dinner in Switzerland, around 18:00 CET, on Wednesday, February 20th, 1991.

We were exhausted.

Conundrum

That hotel, as crappy as it was, was costing us 120 CHF per night, and the only economies we had were the Traveller's Cheques[27] my mum

[22] https://en.wikipedia.org/wiki/Ministro_Pistarini_International_Airport

[23] https://en.wikipedia.org/wiki/Amsterdam_Airport_Schiphol

[24] https://en.wikipedia.org/wiki/Geneva_Airport

[25] https://en.wikipedia.org/wiki/Gen%C3%A8ve-Cornavin_railway_station

[26] https://fr.wikipedia.org/wiki/Les_P%C3%A2quis

[27] https://en.wikipedia.org/wiki/Traveller\XeTeXglyph\numexpr\XeTeXcharglyph"0027\relax{}s_cheque

had bought after selling the appartment. That was the first time I experienced that startup-like “cash burn rate” feeling. We needed to find an appartment to live, a job, you know, a life, and fast.

The following Thursday morning we started visiting real estate offices in Geneva, looking to rent an appartment.

But it turns out my mother could not rent an appartment, because she did not have a job.

And she could not get a job, because she had no fixed address.

See the problem? Oops.

That first weekend in Geneva was dire. My mother broke down in tears from Saturday to Sunday. Even though we had a Swiss passport in our hands, we were stuck in an infinite loop of impossibility.

The following Monday we went to another real estate agency (I think it was Pilet & Renaud[28] but I am not sure), and my mother broke down in front of the person who received us. She explained our broken situation in her broken French.

But this time was different. She looked at us, and said, “please stay here, I have to make a phone call.”

Five minutes later she came back with a paper in her hand. She told us “call this man, his name is Huguenin. He is in charge of the office for Swiss people returning from abroad. I have told him about you, he will be waiting your call.”

Sometimes things happen in the weirdest ways. I do not remember the name of this lady.

Maybe there is something to say about faith here, but I do not know.

“Le Bureau des Suisses de Retour de l’Etranger”

We called right away from a phone booth outside the real estate office. Indeed that Monsieur Huguenin told us to come see him. It turns out there is an office of the Canton of Geneva for Swiss people coming from abroad.

To this day, I do not understand why the Embassy of Buenos Aires did not provide us this phone number to begin with.

Monsieur Huguenin received us that very afternoon. He gave my mother a warranty certificate, and lots of information about where, when, and how to find a job. He told me where and who to ask about my studies (I had to finish high school.)

In less than half an hour, Monsieur Huguenin kicked off our life in Switzerland.

[28] https://www.pilet-renaud.ch/fr/accueil

To Make a Long Story Short

The following Wednesday, February 27th, we moved into a small appartment in Rue Charles-Cusin 10, in the same neighborhood of Pâquis. That evening we celebrated having dinner at a Swiss restaurant in Rue des Pâquis, "L'Auberge de Savièse"[29], close to our new place. Only a week after landing in Geneva, we had a place we could call home.

The following Saturday, March 2nd, Serge Gainsbourg[30] passed away. That is the first public news I remember during those first days in Switzerland. It is weird how memory works sometimes.

By May, my mother had found a job, as a cashier in a local Coop[31] supermarket. I spent my summer at the same Coop, too, working from mid-June to end of August in the warehouses of Satigny[32]. That brought a major boost to my knowledge of French, and some pocket money, too.

By Monday, September 2nd, I started school again. I would graduate from high school with a Swiss "Maturité"[33] in June 1993 in the Collège Sismondi[34], not far from the United Nations building.

A new life started, exactly 30 years ago.

[29] https://www.aubergedesaviese.com/
[30] https://en.wikipedia.org/wiki/Serge_Gainsbourg
[31] https://en.wikipedia.org/wiki/Coop_(Switzerland)
[32] https://en.wikipedia.org/wiki/Satigny
[33] https://en.wikipedia.org/wiki/Matura
[34] https://en.wikipedia.org/wiki/Coll%C3%A8ge_Sismondi

The Agile Invasion

Adrian Kosmaczewski

2021-02-26

Something very big happened during the last 20 years: The Agile Invasion™®©.

A set of white men signed with their names a website called the Agile Manifesto[1], adding a picture of them in front of a whiteboard in 2002. Then lots of people, including me, added their "signature" on the same website, due to a severe case of Distributed Fear Of Missing Out™®© (DFOMO.)

One of them, Ward Cunningham[2], created both the first wiki, and then Extreme Programming[3] in 1996. Another one of the signers, Kent Beck invented Test-Driven Development[4]. Alistair Cockburn[5] invented Crystal. Martin Fowler[6] invented Refactoring[7] and took Ward's idea, mixed it with a blog and turned into a bliki. Andrew Hunt[8] brought pragmatism to the industry. Ken Schwaber[9] and Jeff Sutherland[10] invented Scrum[11].

Not bad for a collective resumé.

Thus a whole new industry was born, and since that moment all developer teams hold a sacrosanct daily standup meeting, even if they still work in noisy open spaces and still struggle to get their software to compile, let alone to run properly.

The Agile Manifesto is an interesting case of a document that has been thoroughly misinterpreted by the whole industry. That must have been, most probably, the reason of its staggering success: let us analyse it a bit.

[1] http://agilemanifesto.org/
[2] https://en.wikipedia.org/wiki/Ward_Cunningham
[3] http://www.extremeprogramming.org/
[4] https://en.wikipedia.org/wiki/Test-driven_development
[5] https://en.wikipedia.org/wiki/Alistair_Cockburn
[6] https://martinfowler.com/
[7] https://www.refactoring.com/
[8] https://en.wikipedia.org/wiki/Andy_Hunt_(author)
[9] https://en.wikipedia.org/wiki/Ken_Schwaber
[10] https://en.wikipedia.org/wiki/Jeff_Sutherland
[11] https://www.scrum.org/

> Individuals and interactions over processes and tools
> Working software over comprehensive documentation
> Customer collaboration over contract negotiation
> Responding to change over following a plan

If you look at the source code on the website of the manifesto, you will see the following:

```
<font size="+3">Individuals and interactions </font><font size="+2">over proces
<font size="+3">Working software </font><font size="+2">over comprehensive docu
<font size="+3">Customer collaboration </font><font size="+2">over contract neg
<font size="+3">Responding to change </font><font size="+2">over following a pl
```

Try not to look at the `<font>` tags for too long, but just enough to see the relative sizes of the text; the +2 and +3 and where they are located. "That is, while there is value in the items on the right, we value the items on the left more," they said. And here is a scoop: based on the font size, we can argue that they value the items on the left 50% more than those on the right.

The problem was, what the average manager understood of this manifesto was completely different:

- There are no tools. You should bring your own device, by the way, because BYOD or forced home office during a pandemic.
- We do not write any more documentation ever. It is actually forbidden. Just provide software that does not crash on start. The rest nobody cares.
- We do not negotiate anything else anymore. It is the way of the CEO or the highway.
- There is no plan to follow. Shut up and code. Pizza, foosball and hackathon! Now you do not have to go home on the weekend anymore.

Oh, and please, be here at 9 AM, we have our standup meeting.

At some point I even became a certified Scrum Master[12] but my certification expired and that link is a 404 now.

Of course, not everything is bad in Agile. There are at least three things I consider to be the most important to ever have come out of the Agile trend.

First, unit tests. The generalization of unit testing in pretty much every programming language on Earth is, in the humble opinion of the author of these words, the most important breakthrough in the history of software engineering. Unit testing brought many side effects to our industry, like new testable architectures, the identity of "legacy code" with untested code, the coupling between refactoring and testing, all of these side effects have made software better.

Second, the concept of "technical debt." Finally MBAs had a metric that could be used in a conversation with developers. Problem is,

[12] https://www.scrumalliance.org/community/profile/akosmaczew

nobody can actually measure it objectively. But that is just a detail. There is, for the first time in the industry, a common concept that both developers and managers can actually use in the same phrase with the same semantics, and this is actually a big thing. The big task now is to actually provide actual meaning to it. But that is another problem.

Third, books like Bertrand Meyer's "Agile! The Good, The Hype And The Ugly"[13] or Steve McConnell's "More Effective Agile"[14] which are the definitive references in the field.

[13] https://deprogrammaticaipsum.com/bertrand-meyer/
[14] https://deprogrammaticaipsum.com/steve-mcconnell/

Opinionated

Adrian Kosmaczewski

2021-03-05

Programming is a very opinionated activity. Unfortunately, those opinions are seldom based on facts, and most of them are futile, and lead to stupid arguments on Reddit or Hacker News or the comments section of a blog.

Most of your conversations with fellow programmers will most probably revolve around one of the subjects enumerated below.

Classical example, the choice of the programming language. Most developers develop a religious attachment to their favourite programming language, and saldly are never able to go past this rather childish feeling. It is like refusing to leave your parents' home and move on with your life once you have the means to.

The truth is, programming languages are just a tool. JavaScript is great at some things, just like C++ is great at others. In some cases you cannot use JavaScript, and you have to use C++; sometimes it is the opposite situation. In some cases you can use several different programming languages to solve the same problem, in more or less the same amount of time, for example Ruby vs. Python. You might like or not the fact that Python uses indentation with specific semantics, but you cannot deny the power of its libraries. Similarly, Ruby is very approachable, but you might dislike that it does not feel fast or efficient.

The truth is that the choice of a particular language is an opinion; and as such, it does not matter. Seriously. Choose any language you want, as long as you and your team are comfortable with it, it has some decent prebuilt libraries ready to use, and it should work out just fine. Just do not enter futile arguments about how your choice is better than someone else's.

There is no "best", not even "better" programming language.

What is "better" is to go and learn a new language every year. I have been consciously been learning a new language every year for the past 30 years, and this has provided me with an invaluable tool: context. Being able to understand how languages are really different from each other, because you have actually tried them, is a gazillion times better than swallowing other people's misconceptions.

Open your mind; learn.

The same argument applies to IDEs, and more generally speaking, to text editors: Visual Studio, Xcode, AppCode, IntelliJ, Eclipse, Sublime Text, Vim, Emacs, Notepad++, EditPlus, Visual Studio Code, TextMate… they are all great and shitty at the same time. Each comes with a long list of relative advantages and an equally long list of terrible flaws that will be never fixed. You read right: never. They are just like that. Take it or leave it.

These tools will be your primary means to earn your bread during your whole career as a code monkey, so it is normal that you pay attention to them. You are going to be standing in front of one of those pieces of software for hours, days, weeks, years, and you are going to both love and hate them. All of them suck and shine in different ways; some are lean; some are heavyweight beasts. Some are fast. Some are slow. Some have awesome debugging capabilities but crash miserably. Some are great in embedded software development and others are better at web development.

And developers will argue, and even worse, they will openly dismiss, criticise, make fun and be arrogant in front of you, just because they have been never able to write anything useful or popular in any of those tools. There is one reason most programmers are so vocal about their preferred tool, and defend it to the end of times: because it is the only one they know how to use properly. As simple as that.

Take a Visual Studio user and put him in front of Vim. Take an Xcode user and try to make him use IntelliJ. Ask a Notepad++ user to switch to Atom. Watch their faces, and listen to the stream of incivilities.

Remember that they are just trying to hide their ignorance by criticising the tool they have in front of them. It is not that the tool is wrong in any way; it is that they just do not know how to use it.

In your case, remember to never, ever talk about something you do not know. Learn, use, try to make something with the tool; then make your own opinion.

The keyboard you choose also defines you somehow as well in the eyes of the “community”, whatever that means. QWERTY vs Dvorak, blue versus cherry MX springs, ergonomic vs. compact, Bluetooth vs. USB, IBM Type M vs Apple Extended II.

The keyboard is the primary means of communication with the computer when you are a developer, and it is primarily used to communicate your ideas inside of the editor we discussed in the previous section. Nothing else. It does not define you in any way, it does not make you better, it does not make you more valuable. But of course a noisy keyboard in an open space office can make people hate you more; that, yes.

We programmers are human beings, and we too are drawn to simpler views of the world, just like anyone else; we are also caught in opinion

storms, and we also enter arguments that have absolutely no value.

You will probably (sadly) find those arguments interesting while you are young, but as soon as years start piling up on you, you will find them futile, and without substance. The important thing is to grow your own taste and knowledge; to try new things, to download new IDEs and languages, to play with them, and also to keep a track record of your discoveries.

Your taste will grow, and also your respect for the choices of others. It is only at that moment that you will be able to evaluate technology properly.

Mentors

Adrian Kosmaczewski

2021-03-12

I have had the tremendous chance of working with incredible programmers through the ages. I will not name them all, but suffice to say that:

- Rami taught me what IDispatch, IUnknown and multiple inheritance were all about;
- Jakob taught me how to find my way in half a million lines of C++ code;
- Damián taught me how to speed up SQL transactions from VBScript;
- Finally, the most important: I learnt from Adam that I was not as bad a developer as I thought I was; and that was a boost to my confidence that stayed to this day.

The names above are not ficticious. They will recognise themselves, hopefully.

I also had the tremendous chance of working with state-of-the-art assholes, who also taught me a lot. I will not name them; they are not worth our time.

What is important is learning to separate the wheat from the chaff; recognise, at every moment of your path through this industry, who is there to help you and who is there to fuck your career up.

Teaching

Adrian Kosmaczewski

2021-03-19

In November 2015 I attended the DO iOS[1] conference in Amsterdam. After the event was over, I had dinner in a pub with my friend Daniel Steinberg[2], whom I had not met in more than three years, and who had just held a terrific Swift workshop in the event.

I remember during that evening, Daniel mentioned one of the most prominent laws of learning ever muttered, one that probably was first pronounced by Plato, Archimedes or some other wise Greek person 3 thousand years ago:

> Teaching is the best way to learn.

Many programmers are usually very shy of talking in public, let alone capable of explaining technical concepts in intelligible ways to non-technical audiences.

Achieving both is a worthy goal, and one that can paradoxically teach you a lot.

Speaking

The most important soft skill you should master is presenting in front of an audience. It is surprising, but I still come across many MBA and university curricula without any course about presentation skills as a core requirement.

The thing is, you are going to be showing your work to people all the time, whether you want it or not; hence, you should start working out your presentation skills as soon as you can.

How? Well, very simple; go to the local user group of your technology of choice, and give a ten minute talk about any subject you like. Just that. Did I say ten? Heck no, just five minutes is enough. And make sure that there are no more than ten people in that session. Small audience, small subject, short time. That is all.

Start small. Smile and breathe.

[1] https://do-ios.com/

[2] https://dimsumthinking.com/

Sounds easy? Well, there is a catch: you should write down your speech. In Markdown or in Asciidoc (seriously, avoid Word and company). This is the most important part; to prepare your talk. It is fundamental.

It does not matter if there are ten or a thousand people in the room, you have to know your subject inside out, and for that, the best way is to write down your speech. It will help you remember it, it will reduce your anxiety (because, yes, you will be anxious) and it will increase your confidence.

By the way, written speeches can also make for terrific blog posts. Just saying[3].

Where to start? Very simply; write down what you want to say. Make sure that you have an introduction, a body, and a conclusion: remember the simple formula for any good text:

1. Say what you are going to talk about.
2. Talk.
3. Summarise and repeat what you just talked about.

Unlike programming, where we strive to DRY[4], repetition is very important in speeches. Getting your point across means repeating yourself.

At the end, thank everyone, and ask politely for questions. Smile and breathe. It is done.

Once you have your speech on paper, rehearse. Three or four times, at least. With a timer. You can use your phone for that, or a kitchen timer, it does not matter. Preferably in front of a non-technical audience, like your partner or some best friend. Ask them how you did and remember to smile and breathe.

Take time to breathe. It is super important.

Then, after you have written your speech, and only then, you should create your slides. Make them eye-catchy, not a lot of text, and with big fonts. Use images. Lots of them. Images are better than text.

And then, give your presentation. Remember, smile and breathe.

Thank everyone and ask politely for questions. Smile and breathe one last time. It is done.

If you liked presenting to ten people a five minute subject, you should gradually expand both variables. Pitch your talk to technical conferences – they crave new content, and many find it very difficult to find good speakers.

After ten or twelve times of doing that, one day you will be talking for a whole hour in front of a thousand people. It happened to me and it is frankly exhilarating.

[3] /tags/speeches/

[4] https://en.wikipedia.org/wiki/Don%27t_repeat_yourself

Teaching

The next logical step will be to use the material you wrote down (in Markdown or Asciidoc, remember), and expand on it. Think about a small group of twelve to twenty people, to which you could teach that during a whole day.

It is a business! If you charge 200 USD to each person for such a workshop (a relatively reasonable price), and you have twenty people signing up, you can make four grand a day. Renting a classroom might cost you from 300 to 1000 USD per day, depending on location and whether you offer food or not. In any case, repeat that operation one week a month and you have a very decent salary. Use the rest of the month to create new trainings, rinse and repeat.

Publishing

This is the final step. Take those classroom notes you wrote down, and publish a book. Yes, do it.

Because you have written your notes in Markdown or Asciidoc (you have, right?), you can use pandoc[5] to export them to a beautiful PDF, EPUB, HTML and a few more formats in one shot.

Next thing you know you can start pitching a full manuscript to O'Reilly or Pragmatic Programmers, and maybe you will get yourself published! Publishers love when you pitch an almost-ready book, because it makes them more confident of your capacity to deliver stuff in time. Do not worry about being copied; the ones I mentioned above are trustworthy.

And if you do not get a publishing contract with one of those publishers, then you can go to Leanpub.com and (again, thanks to Markdown) compile your notes and publish your book anyway. Just do it.

Coaching

Finally, at some point you are going to find a younger developer hungry for knowledge and anecdotes. Coach them. Use your knowledge to shape a better generation of developers in the future. Show them how you used to code before, and let them show you new tricks.

Coaching a younger developer should be mandatory for all of us; it makes coachees wiser, and it keeps coaches younger at heart.

[5] https://pandoc.org/

Pascal

Adrian Kosmaczewski

2021-03-26

From 2013 to 2019 I lived in a small town, thirty minutes north of Zürich by train, called Schaffhausen[1]. Where I lived, our neighbors organised every year a gathering, with traditional food and drinks, and where everybody talked to me in Swiss German[2]. Even if I did not always get what was going on, they were so friendly I could not stop smiling.

Smiling is the key for happiness and to have a nice relationship with your neighbors.

So it turns out that after a few years in Schaffhausen my German skills were improving, in spite of all my efforts for that not to happen, and in one of those gatherings I had a nice conversation – at least that was my impression – with one of my neighbors.

They all knew that I "worked with computers", whatever that meant, and I helped some of them reinstall the occasional printer driver here or there. This is part of our Karma.

So during this conversation with this neighbor, I was told he knew a professor from ETHZ, the Zurich Federal Polytechnic Institute where Einstein studied.

He told me that they went together to primary school in a city between Schaffhausen and Zurich called Winterthur[3].

He told me that they met every so often for dinner, and that next time he would gladly introduce him to me.

He tells me that I might know his name.

He tells me his friend's name is Niklaus Wirth[4].

You can imagine my reaction.

Pascal[5] was the second programming language I learnt, back in 1993. My best friend in high school, Bertrand, was an Amiga[6] wizard who

[1] https://en.wikipedia.org/wiki/Schaffhausen
[2] https://en.wikipedia.org/wiki/Swiss_German
[3] https://en.wikipedia.org/wiki/Winterthur
[4] https://en.wikipedia.org/wiki/Niklaus_Wirth
[5] https://en.wikipedia.org/wiki/Pascal_(programming_language)
[6] https://en.wikipedia.org/wiki/Amiga

taught me a lot about computers in the beginning.

(Speaking about Amiga wizards, I enjoy watching Graham and Steve[7] talking about the Amiga on Twitch.)

Following Bertrand's tips, I first learnt Turbo Pascal[8], then Object Pascal, then Delphi, and I also tried to play with Oberon[9] but without much success. Pascal is a very, very Swiss language. Everything has its place. Everything is defined, typed, and very easy to understand. Everything is verbose.

Bertrand passed away almost five years ago. I miss him a lot.

As for my neighbor, well, I never met Dr. Wirth, at least not so far. I would love to.

[7] https://www.dosamigans.com/

[8] https://en.wikipedia.org/wiki/Turbo_Pascal

[9] https://en.wikipedia.org/wiki/Oberon_(programming_language)

Growing as a Developer

Adrian Kosmaczewski

2021-04-02

You will outlive IDEs. At some point you'll find them bloated, slow, clunky. That's good. But there are moments when using an IDE is still better than not using one. You will learn to distinguish when that happens.

Don't fall for the noise of hype. Most of what you see and use every day has been already invented a thousand times. But listen to the signal of hype and learn to separate chaff from grain.

We work in an industry that does not value wisdom. Most of your managers will not understand a word you say, no matter how many years of experience you have, yet they will have the power to fire you anyway. Live with it. Strive to find a place where you will be heard. At 35 you will be old already for this industry, sadly.

If you do not touch type by the age of 30, make sure you learn it. It's going to help you a lot.

You should coach somebody. It makes the world a better place.

Exercise. Seriously, sitting in front of a computer all day brings back pain, headaches, overall fatigue, and it makes you grumpy. Standing desks are good but not enough. Get a treadmill or an elliptical bike and exercise, at least 30 minutes every day.

Programming jobs pay well. But don't take a job because of the money. It's a decision that sooner or later backfires on you, and you're going to regret it.

Planning

Adrian Kosmaczewski

2021-04-09

When James Patterson, one of the best-selling authors in the world, had the idea for “Honeymoon,” his 2005 novel, he started writing an outline. He iterated 5 or 6 different times until he got the right flow of the story, the right characters, the right bad guys, and the right conclusion. He showed the final version of his outline to publishers, and after some modifications he started writing the novel itself.

James used his outline as a reference, whenever he got lost in the writing of his novel, to find the right path and to help him move the story forward.

When Quentin Tarantino, one of the most popular movie directors in Hollywood, wanted to create “Pulp Fiction,” he wrote a screenplay. He went through several iterations, each time changing the story a little bit, until he found the right characters, the right gags, the right bad guys, and the right conclusion. He showed the final version of his outline to studios, and after some back and forth he wrote the shooting screenplay and then directed, edited and released the film itself.

Quentin and his actors used the screenplay as a reference both during shooting and edition, to make sure that the dialogs were right, that the scenes were compelling, and that the final edition conveyed the right message.

When software developers start a project, they usually… open their favorite IDE, click on the `File / New project…` menu entry, and start coding right away. In the old days, some iOS developer teams took the time to create the famous “Application Definition Statement,” as it was recommended in the Apple Mobile Design Guidelines around 2009. Some other teams might have a project leader with a clear vision of the product being created.

But unfortunately, most development teams wander in the dark, in search of the mythical “MVP” (Minimum Viable Product.) The one that some marketing wizard (unfortunately not yet a member of this promising young startup) would (could?) eventually sell to somebody else in order to bootstrap the whole thing.

See the pattern?

COVID-19 Vaccination Progress

Adrian Kosmaczewski

2021-04-16

The (low) speed of the COVID-19 vaccination in Switzerland is infuriating. The current graphs[1] show almost a linear progression. Linear, not quadratic, not exponential. Linear. Fucking linear. They say it's going to accelerate from now on, well, we'll see about that.

I extrapolated the official data to find out the possible date in which all of Switzerland will have had both doses. Of course I know that many don't want it – an idiotic reality, if you ask me, but whatever. Vaccines should be mandatory and basta.

The Python script is as simple and as stupid as you can imagine, and here it is[2] for your use and verification. I used Python because of SciPy[3] and NumPy[4], awesome stuff that allows you to make great stats with just a few lines of code.

Thanks to the Swiss Federal government for the data, have fun, and stay safe.

Update, 2021-04-23: Thanks to Alexandre Joly[5] there's now a JavaScript version[6] compatible with Scriptable[7], an iOS application that allows to create custom home screen widgets.

Update, 2021-05-14: The evaluation of current data indicates an acceleration[8] of the vaccination rate; 72.5% since May 1st, up to 18'200 fully vaccinated people/day from just around 10'000 before that date.

[1] https://www.covid19.admin.ch/en/
[2] https://gitlab.com/akosma/covid-19-vaccination-in-switzerland
[3] https://www.scipy.org/
[4] https://numpy.org/
[5] https://twitter.com/jolyAlexandre
[6] https://gist.github.com/mekanics/b6ecb764d5b5048fd623767454ddca6b
[7] https://scriptable.app/
[8] https://twitter.com/akosma/status/1392131810755289088

Douglas Crockford

Adrian Kosmaczewski

2021-04-23

Around 2002 I found a small website created by a certain Douglas Crockford, in which he claimed that JavaScript was the World's Most Misunderstood Programming Language[1]. It was intriguing, so I read, and I discovered that there was much more to the language than I thought at first glance.

> JavaScript's C-like syntax, including curly braces and the clunky for statement, makes it appear to be an ordinary procedural language. This is misleading because JavaScript has more in common with functional languages like Lisp or Scheme than with C or Java. It has arrays instead of lists and objects instead of property lists. Functions are first class. It has closures. You get lambdas without having to balance all those parens.

Back then I had no idea that there was a family of languages called "functional;" actually the name seemed odd to me because, well, Pascal had functions and apparently it was not functional, so I was quite confused.

Douglas is also the author of "JavaScript: The Good Parts"[2] book, which had quite an impact back in the day. It is probably one of the best sellers of our industry. The book has always been sarcastically compared to its fatter sibling, the venerable "JavaScript: The Definitive Guide"[3] by David Flanagan, which has been around since 1996 and whose latest edition, the seventh, came out last year.

Back in 2002, when I found Douglas' writing on the web for the first time, I said that he was probably the only person that actually understood JavaScript correctly. I think I was right.

---

[1] http://www.crockford.com/javascript/javascript.html

[2] https://www.amazon.com/JavaScript-Good-Parts-Douglas-Crockford/dp/0596517742/

[3] https://www.amazon.com/JavaScript-Definitive-Guide-Activate-Guides/dp/0596805527

Ham Is to Hamsters

Adrian Kosmaczewski

2021-04-30

I published my first web page in the server of my university[1]. It was at the end of August, 1996. It will be 25 years soon. I was about to turn 23, and I had taught myself HTML with Liz Castro's excellent book "HTML for the World Wide Web"[2].

At some point in my explorations of the World Wide web I came across a dynamic web page. And by dynamic I mean an `alert()` dialog box. The good old ones generated by Netscape 2[3] running on Windows 95[4].

An alert box. This was no ordinary web page, I told myself, and when I selected the venerable "View Source" option on the menus, I saw for the first time a snippet a curly-bracketed programming language called JavaScript.

It could not do much more than validating forms back then. That was the primary use of those <SCRIPT> tags (yes, they were all uppercase back then). So you would add some JavaScript here and there, and if the fields of a form were empty, bam! The alert would appear. This allowed us, obnoxious nerds, to annoy the few people daring to install Netscape in their computers and to redirect them forever out of our websites.

Usability has never been a strong characteristic of the developer community.

[1] http://sc2a.unige.ch/~kosmacze

[2] https://www.amazon.de/Html-World-Visual-Quickstart-Guide/dp/020168862X

[3] https://en.wikipedia.org/wiki/Netscape_(web_browser)

[4] https://en.wikipedia.org/wiki/Windows_95

Specialized Generic Methods in Rust

Adrian Kosmaczewski

2021-05-07

I have not been so interested in a language in years as I am right now with Rust[1]. Readers of De Programmatica Ipsum[2] have probably sensed this last Monday when we published "The Great Rewriting in Rust"[3], which has been a hit article this week.

To say that I'm fascinated by Rust is an understatement.

So, as an exercise, I wanted to replicate in Rust one of the subsystems I used for my master thesis project[4] back in 2008. That one was a C++03 project; back then C++11 was still in the works, so I could not use many new features that are nowadays common in the standard, like std::tuple[5] and others.

That project had a homemade ORM, basically implementing all objects as `std::map` of properties using the `Poco::Any` object. Those "properties" were just generic key/value pairs, and I got the inspiration from a C++ book I had read for work, "Financial Instrument Pricing Using C++" by Daniel Duffy (book which has since been updated to C++11). The final goal of that ORM was to save a complete object graph into a SQLite file.

Here is the original[6] code for the implementation of the property pattern I used in my project.

So, how would we do this in Rust? I started with the most naive implementation, and to my surprise (given my rookie level of Rust knowledge) it just worked:

```
struct Property<T> {
    name: String,
    value: T,
}
```

I added a convenience function attached to that struct, just to have a nicer way to create new instances in my code:

[1] https://www.rust-lang.org/
[2] https://deprogrammaticaipsum.com/
[3] https://deprogrammaticaipsum.com/the-great-rewriting-in-rust/
[4] https://github.com/akosma/remproject/
[5] https://en.cppreference.com/w/cpp/utility/tuple
[6] https://github.com/akosma/remproject/blob/master/src/utility/Property.h

```
impl<T> Property<T> {
    fn create(name: &str, value: T) -> Property<T> {
        Property {
            name: String::from(name),
            value: value,
        }
    }
}
```

Now I needed to create a bag structure holding lots of these things:

```
struct PropertyMap<T> {
    pub values: HashMap<String, Property<T>>,
}
```

And of course I added a set of convenience functions to help me use this type:

```
impl<T> PropertyMap<T> {
    fn create(size: usize, properties: Vec<Property<T>>) -> PropertyMap<T> {
        let mut values: HashMap<String, Property<T>> = HashMap::with_capacity(s
        for property in properties {
            let name = property.name.clone();
            values.insert(name, property);
        }
        PropertyMap { values: values }
    }

    fn get(&self, key: &str) -> Result<&Property<T>, String> {
        if self.values.contains_key(key) {
            return Ok(&self.values[key]);
        }
        return Err(format!("Value '{}' not found", key));
    }
}
```

Beautiful implementation of `get()` with the very handy `Result` type. Of course, now I need a way to pretty-print these things in the console:

```
impl<T> fmt::Display for Property<T>
where
    T: fmt::Display,
{
    fn fmt(&self, f: &mut fmt::Formatter) -> fmt::Result {
        write!(f, "{} = {}", self.name, self.value)
    }
}

impl<T> fmt::Display for PropertyMap<T>
where
    T: fmt::Display,
{
    fn fmt(&self, f: &mut fmt::Formatter) -> fmt::Result {
```

```
        for (key, value) in &(self.values) {
            writeln!(f, "{} => {}", key, value)?;
        }
        return Ok(());
    }
}
```

So far, so good. Now I can start playing with those things:

```
fn main() {
    let a = Property::create("age", 22);

    // Create map and print it
    let map: PropertyMap<i32> =
        PropertyMap::create(30, vec![a.clone()]);
    println!("Full map =>");
    println!("{}", map);

    // Try to find item in map
    for key in ["tata", "toti", "age"].iter() {
        let found = map.get(key);
        match found {
            Ok(value) => println!("FOUND '{}' => {}", key, value),
            Err(message) => println!("{}", message),
        }
    }

    // Create properties of other types
    let b = Property::create("name", "Roger");
    let c = Property::create("valid", true);
    let d = Property::create("cost", 6666.25);
}
```

Looking good! Now I also implemented equality, so that I can compare properties with one another:

```
impl<T> cmp::Eq for Property<T> where T: cmp::PartialEq {}

impl<T> cmp::PartialEq for Property<T>
where
    T: cmp::PartialEq,
{
    fn eq(&self, other: &Property<T>) -> bool {
        return self.name == other.name && self.value == other.value;
    }
}

fn main() {
    let a = Property::create("age", 22);
    let a1 = a.clone();
    let a2 = Property::create("age", 256);
    let a3 = Property::create("whatever", 22);
```

```
    println!("{}", a == a1); // true
    println!("{}", a == a2); // false
    println!("{}", a == a3); // false
}
```

Of course, it does not make sense to compare properties that are not of the same type; the compiler would complain, anyway.

Now I wanted to push things a bit further; when creating SQL statements for inserting or updating, we need to surround strings with single quotes; my first reaction was to create a generic `format_for_database()` function, and then provide a specialized version for `Property<String>` and `Property<&'static str>`, but I quickly found out that this is currently not possible in Rust.

So I ended up doing the following instead:

```
type IntProperty = Property<i32>;
type FloatProperty = Property<f64>;
type BoolProperty = Property<bool>;
type StringProperty = Property<&'static str>;

impl IntProperty {
    fn format_for_database(&self) -> String {
        format!("{} = {}", self.name, self.value)
    }
}

impl BoolProperty {
    fn format_for_database(&self) -> String {
        let val = if self.value { "TRUE" } else { "FALSE" };
        format!("{} = {}", self.name, val)
    }
}

impl FloatProperty {
    fn format_for_database(&self) -> String {
        format!("{} = {}", self.name, self.value)
    }
}

impl StringProperty {
    fn format_for_database(&self) -> String {
        format!("{} = '{}'", self.name, self.value)
    }
}
```

Now we can do very polymorphic calls like this:

```
fn main() {
    let a = Property::create("age", 22);
    println!("Property a => {}", a.format_for_database());
```

```
    let b = Property::create("name", "Roger");
    println!("{}", b.format_for_database());

    let c = Property::create("valid", true);
    println!("{}", c.format_for_database());

    let d = Property::create("cost", 6666.25);
    println!("{}", d.format_for_database());
}
```

Verbose, but gets things done, waiting until the language supports such feature. Ideally I would have implemented a generic `format_for_database()` function in the generic `impl<T>` block, but if I do that, I get error `E0592`, complaining about a duplicate definition.

You can find the final code in my GitLab account[7]. The next step for this project will be to use Rust's own Any trait[8] and translate the original C++ code into a working Rust version.

[7] https://gitlab.com/akosma/generic-properties-in-rust
[8] https://doc.rust-lang.org/std/any/trait.Any.html

Generic Enum Type in Rust

Adrian Kosmaczewski

2021-05-14

I continue my exploration of Rust through a simple implementation of the Active Record design pattern.

Last week[1] I told the story of how I had created a simple `Property<T>` type, and a collection thereof. But of course a real object has many properties, and of various different types at once; so such a `PropertyMap<T>` was not exactly what I needed, unless all of your properties are the same type.

To solve that issue, in my original C++ implementation from 2008[2] I used the Poco::Any[3] type, itself based on the original Boost library[4] of the same name, created by Kevlin Henney[5].

I quickly found out that Rust had an Any trait[6], but after some fiddling I could not make it work the way I wanted.

That had to do more with my lack of knowledge of Rust than anything else; see, Rust's Any is a trait, not a struct, so the semantics in my head were, of course, twisted, and I could not find a solution.

Thankfully DuckDuckGo was kind enough to provide me with the answer, in the form of a blog post by Simone Vittori[7] published last year. The idea is simple, and will sound familiar to those using Swift, by the way: use an `enum`.

```rust
#[derive(Clone, Debug)]
pub enum AnyProperty {
    Str(Property<&'static str>),
    Int(Property<i32>),
    Float(Property<f64>),
    Bool(Property<bool>),
    DateTime(Property<DateTime<Utc>>),
}
```

[1] /blog/specialized-generic-methods-in-rust/
[2] https://github.com/akosma/remproject/blob/master/src/storage/AnyProperty.h
[3] https://pocoproject.org/docs/Poco.Any.html
[4] https://www.boost.org/doc/libs/1_61_0/doc/html/any.html
[5] https://en.wikipedia.org/wiki/Kevlin_Henney
[6] https://doc.rust-lang.org/std/any/trait.Any.html
[7] https://www.simonewebdesign.it/rust-hashmap-insert-values-multiple-types/

Now I can hold many of these in the same bag:

```
#[derive(Clone, Debug)]
pub struct AnyPropertyMap {
    properties: HashMap<String, AnyProperty>,
}
```

The whole API is very simple to use now:

```
#[test]
fn create_any_property_map() {
    let a = AnyProperty::from_int("int", 22);
    let b = AnyProperty::from_str("str", "value");
    let c = AnyProperty::from_float("float", 1.234);
    let d = AnyProperty::from_bool("bool", true);
    let e = AnyProperty::from_datetime("date", Utc::now());
    let m = AnyPropertyMap::from(vec![a, b, c, d, e]);
    assert_eq!(m.len(), 5);
}
```

And thanks to pattern matching, translating such properties to strings becomes very simple and straighforward. This will look familiar to Swift developers; the influence of Rust in Swift is here more visible than anywhere else, in my opinion.

```
pub fn get_value(&self) -> String {
    return match self {
        AnyProperty::Str(prop) => format!("'{}'", prop.value),
        AnyProperty::Int(prop) => format!("{}", prop.value),
        AnyProperty::Float(prop) => format!("{}", prop.value),
        AnyProperty::Bool(prop) => {
            let val = if prop.value { "TRUE" } else { "FALSE" };
            format!("{}", val)
        }
        AnyProperty::DateTime(prop) => format!("'{}'", prop.value.to_rfc3339())
    };
}
```

To be able to handle date and time information, I chose the chrono[8] crate, which provides a very handy and useful DateTime struct[9] to work with.

I started working on the actual ActiveRecord pattern implementation, and I have published the code as usual in my GitLab[10] account. Feel free to clone and play with it. The code is already bundled with as many unit tests[11] as I could fit, and more will come. The usual `cargo test` command will execute them for your testing pleasure.

[8] https://docs.rs/crate/chrono/0.4.19
[9] https://docs.rs/chrono/0.4.19/chrono/struct.DateTime.html
[10] https://gitlab.com/akosma/rust-active-record
[11] https://gitlab.com/akosma/rust-active-record/-/blob/master/src/lib.rs

The final objective of this exercise is to achieve an API that will look more or less like this:

```
let mut o = Object::create();
o.set_name("Milk");
o.set_price(1.23);
o.set_valid(true);
o.save();

o.set_price(2.56);
o.set_valid(false);
o.save();
```

And at the end, a SQLite file will be generated with those objects stored in a table. I am planning to explore Rust's macro system and syntax in order to generate most of the boilerplate code required for this library to be useful.

I know that many consider Active Record an antipattern, but I do not pretend this library to become a production thing anytime soon; of course, the Internet being what it is, I added a BSD 3 license to the bundle, just in case.

Upcoming Explorations

Another thing I started exploring this week was how to create web applications with Rust; in particular I built a simple web application using Actix[12] with the Askama[13] template library, and then wrapping the whole thing inside a Docker container image.

But I will leave that for a future post. In short, I loved it; the final code is ultra fast, in the shape of a very small self-contained binary, perfectly suitable for a deployment in Kubernetes. I rewrote a very simple Flask[14] application in a few hours. More soon!

[12] https://actix.rs/
[13] https://djc.github.io/askama/
[14] https://flask.palletsprojects.com/

First Web App in Rust

Adrian Kosmaczewski

2021-05-21

My exploration of Rust continues; this week, I rewrote a Python Flask application I use for demos at work.

Python Application

The original app is a simple Flask[1] application, running with Python 3.7, which just outputs a random number and a funny message when you use it. The funny message comes from the fortune[2] program, which must be installed in the host.

Here is the code:

```
import os
from flask import Flask, request, Response, jsonify
from flask.templating import render_template
from subprocess import run, PIPE
from random import randrange

app = Flask(__name__)

version = '1.0'

def get_fortune():
    number = randrange(1000)
    fortune = run('fortune', stdout=PIPE, text=True).stdout
    return number, fortune

@app.route("/")
def fortune():
    number, fortune = get_fortune()
    mimetype = request.mimetype
    if mimetype == 'application/json':
        resp = jsonify({ 'number': number,
                         'fortune': fortune,
                         'version': version })
        return resp
```

[1] https://flask.palletsprojects.com/

[2] https://en.wikipedia.org/wiki/Fortune_%28Unix%29

```
    if mimetype == 'text/plain':
        result = 'Fortune %s cookie of the day #%s:\n\n%s' % (version, str(numb
        resp = Response(result, mimetype='text/plain')
        return resp

    html = render_template('fortune.html', number=number, fortune=fortune, vers
    resp = Response(html, mimetype='text/html')
    return resp

if __name__ == "__main__":
    app.run(host='0.0.0.0', port=os.environ.get('listenport', 9090))
```

There's just one endpoint, and the whole thing listens on port 9090. Depending on the `Content-Type` header of your request, the app spits out JSON, plain text, or just good old HTML, generated using Jinja.

Let's Get Rusty

After learning about Rust web frameworks, here's my translation of the application above using the Actix[3] crate, together with the Askama[4] template library.

```
use actix_web::{get, App, HttpRequest, HttpResponse, HttpServer, Responder};
use askama::Template;
use rand::Rng;
use serde::{Deserialize, Serialize};
use std::process::Command;

#[derive(Template)]
#[template(path = "fortune.html")]
#[derive(Serialize, Deserialize)]
struct FortuneData {
    number: u32,
    fortune: String,
    version: String,
}

impl FortuneData {
    fn new() -> FortuneData {
        let mut rng = rand::thread_rng();
        let command = Command::new("fortune")
            .output()
            .expect("failed to execute process");
        FortuneData {
            fortune: format!("{}", String::from_utf8_lossy(&command.stdout)),
            number: rng.gen_range(0..1000),
            version: String::from("1.0-rust"),
```

[3] https://actix.rs/
[4] https://djc.github.io/askama/

```
        }
    }

    fn to_plain(&self) -> String {
        format!(
            "Fortune {} cookie of the day #{}:\n\n{}",
            self.version, self.number, self.fortune
        )
    }
}

fn get_content_type<'a>(req: &'a HttpRequest) -> Option<&'a str> {
    req.headers().get("Content-Type")?.to_str().ok()
}

#[get("/")]
async fn fortune(req: HttpRequest) -> impl Responder {
    let response = FortuneData::new();
    match get_content_type(&req) {
        Some(content_type) => {
            if content_type == "text/plain" {
                return HttpResponse::Ok().body(response.to_plain());
            }
            if content_type == "application/json" {
                return HttpResponse::Ok().json(response);
            }
        }
        None => {}
    }
    HttpResponse::Ok().body(response.render().unwrap())
}

#[actix_web::main]
async fn main() -> std::io::Result<()> {
    HttpServer::new(|| App::new().service(fortune))
        .bind("0.0.0.0:9090")?
        .run()
        .await
}
```

This Rust application does exactly the same as the previous one in Python, and both can be stored in a nice container, to be executed in your nearest Kubernetes cluster.

The HTML Template

The nice thing of this rewriting was that I could reuse the same `struct FortuneData` for creating JSON, and also to feed the Askama template. Very handy and very elegant.

This is, by the way, the HTML template used by both applications, reused without changes from one app to the other, and bearing the filename `templates/fortune.html` in both cases.

```
<html>

<head>
    <title>FORTUNE COOKIE</title>

    <style>
        * {
            color: green;
            font-family: Cambria, Cochin, Georgia, Times, 'Times New Roman', se
            font-size: x-large;
        }

        #main {
            width: 80%;
        }
    </style>
</head>

<body>
    <div id="main">
        <h1>Fortune cookie of the day #{{ number }}</h1>

        <p>{{ fortune }}</p>

        <hr>

        <p>Version {{ version }}</p>
    </div>
</body>

</html>
```

Container Image and Dockerfile

The main difference between both apps, beyond the obvious syntactic differences, had to do with the creation of container images and the required `Dockerfile`. While in the case of the Python app, the build process was short and the resulting image was huge, in the case of Rust the build process was awfully long, and the resulting image was subatomically small. Talk about tradeoffs.

To reduce the build times of the container image, I used the trick shown in this blog post[5] which consists of having two separate `cargo build` commands; the first one installs the dependencies, which seldom change, while the second one actually compiles the executable

[5] https://blog.logrocket.com/packaging-a-rust-web-service-using-docker/

from the main Rust file, which of course changes more often.

Let's compare the Dockerfiles, both using Alpine as a basis. First, the Python version:

```
FROM python:3.7-alpine
RUN apk add fortune
WORKDIR /usr/src/app
COPY requirements.txt ./
RUN pip install --no-cache-dir -r requirements.txt
COPY app.py /usr/src/app
COPY templates /usr/src/app/templates/
USER 1001
EXPOSE 9090
CMD [ "python", "app.py" ]
```

Simple and sweet. Of course, the Python runtime increases the size of the final image quite dramatically.

And now the Rust one, itself using a multi-step approach:

```
# Step 1: builder image
# We're using musl to generate smaller images based on Alpine Linux
# Base image: https://github.com/emk/rust-musl-builder
FROM ekidd/rust-musl-builder:stable as builder

# This Dockerfile contains two `cargo build` commands;
# The first compiles the dependencies of the application
# while the second one compiles and links the app itself, which
# changes less often than the dependencies.
# Inspired from
# https://blog.logrocket.com/packaging-a-rust-web-service-using-docker/

# Compile dependencies
RUN USER=root cargo new --bin rust-fortune
WORKDIR ./rust-fortune
COPY ./Cargo.lock ./Cargo.lock
COPY ./Cargo.toml ./Cargo.toml
RUN cargo build --release

# Compile the app itself
RUN rm src/*.rs
RUN rm ./target/x86_64-unknown-linux-musl/release/deps/rust_fortune*
ADD src ./src
ADD templates ./templates
RUN cargo build --release

# Step 2: runtime image
FROM alpine:latest
EXPOSE 9090

# Add dependencies
```

```
ENV TZ=Etc/UTC
RUN apk update \
    && apk add --no-cache fortune ca-certificates tzdata \
    && rm -rf /var/cache/apk/*

# Copy self-contained executable
COPY --from=builder /home/rust/src/rust-fortune/target/x86_64-unknown-linux-mus

# Never run as root
USER 1001:0
CMD ["/usr/local/bin/rust-fortune"]
```

The final result is a super small image, holding a single executable that contains everything it needs to run; even the HTML template is embedded inside of the application, and you don't need to copy it separately. Of course, containers based on this image start and stop super fast, and consume less memory than the other.

So, as I said; your tradeoff is a bit longer build time for less memory consumption and faster execution. I'd say it's totally worth it, particularly if your final image is created in a CI/CD system such as a GitLab pipeline, where you could not care less about building times.

A Compilation of Old Blog Posts

Adrian Kosmaczewski

2021-05-28

Looking inside some old archives I came across my first attempt at a book. Back in 2009 I compiled many of the articles I had published in my first blog, which I had started in 2004, and generated a PDF with them.

It was certainly amusing and enlightening to rediscover the things that tickled my curiosity back then. If you haven't anything else better to do, feel free to read it[1].

[1] /books/Open_Kosmaczewski_Book/Open_Kosmaczewski.pdf

Vaccines and Software Developers

Adrian Kosmaczewski

2021-06-04

I am a bit tired and disappointed by the attitude of many in our industry, people supposedly eager to test new technologies and "move the human race forward". Yes, just like the Apple ads of the 90s campaign "Think different" used to say.

Many (not all, thankfully) of those same people, eager to quote Steve Jobs and to test every single JavaScript framework under the Sun, are very quick to write me (in private, of course) about how me telling people to get vaccinated is an act of politics that has no place in Switzerland.

These are Swiss people, if you need to know. Switzerland has chosen not to make vaccination mandatory, and as a result lots of people are actually campaigning to discredit the vaccine campaign, or the vaccines themselves, in any possible way. Yes, here in Switzerland.

Let me be very clear. Choosing not to be vaccinated in Switzerland is legal, yes; but it is an immoral, repugnant, idiotic, and utterly disgusting act.

For context, many in the country where I was born would give all their possessions to have a single shot of a Pfizer or Moderna jabs, instead of the Russian or Chinese vaccines they can only afford to have, and even so, only if you're a well-placed politician or a celebrity.

But no, of course not, there are Swiss people who have the choice to get vaccinated and help the whole community to get better, not only for themselves, but for all of us, because vaccinated people prevent the virus from reproducing further, moving along chains of infection. But no, they won't, because freedom.

What a twisted, sick sense of freedom, indeed. While at the same time climate activists are arrested and interrogated[1] in ways that aren't worthy of a country where the Red Cross has its headquarters.

Those who campaign against the vaccine are the same people (in many cases SVP/UDC activists, who are the cancer of this country) silencing voices who dare speak against the untouchable Swiss Army or some other Swiss "value" – clearly, freedom of expression ain't one.

[1] https://www.rts.ch/info/suisse/12246860-perquisitions-a-la-greve-du-climat-vaud-apres-un-appel-a-la-greve-militaire.html

Thankfully, the numbers[2] confirm that already 40% of the population has chosen to be vaccinated, and have received at least one dosis; and that the rate of vaccination[3] is slightly growing week after week, reaching now an impressive number of 42'000 people fully vaccinated per day – that is, with two doses.

Even better: more than 70% of all people above 70 years old have already received both dosis. This is outstanding news, no matter how you look at them.

So I trust that the wisdom of the majority of the Swiss people will bring in more volunteers to the vaccination centers in weeks to come.

Getting vaccinated is a duty. If you think otherwise, please keep your thoughts to yourself and don't waste anyone's time.

I call all software developers, all DevOps engineers, all database administrators, all Agile coaches in this country, all software industry workers, to not only get vaccinated, but to make your voices heard, and to tell people to get vaccinated.

It is only by making our voices louder than we can block that outrageous noise caused by the ignorant mob of antivaxxers polluting this country.

This includes the most common and inane of criticism, which is the one stating that this vaccine could have never be developed so quickly. It is 2021 people. Vaccines are developed in months, yet your shitty full-stack application still doesn't run properly after years of work.

Software workers could learn a thing or two from vaccine engineers, if you ask me.

[2] https://www.covid19.admin.ch/en/epidemiologic/vacc-persons
[3] https://gitlab.com/akosma/covid-19-vaccination-in-switzerland

The End of the Tunnel?

Adrian Kosmaczewski

2021-06-11

From a Swiss perspective it would be easy to think that the COVID-19 crisis is reaching its end. That's a rather myopic point of view; the truth is that many places in the world are going through the worst patches at this point.

Take for example my place of birth, Argentina. If you follow the news, you know that at the moment it is one of the countries in the world with the highest numbers of new daily cases and deaths.

The vaccination situation is patchy, at best. My father has not received his first dosis yet; even though many younger people are reporting on social media that they got it. The organization of the whole campaign is an absolute joke; but we knew that already. It is the Argentine style of organization. No surprises here, sadly.

On the other side of the equation, I write these lines from the confort of a Swiss home; my wife is fully vaccinated, I have had my first injection last week, and the weather is finally worthy of a Summer season.

It would be foolish to see the typical Swiss landscape of organization and peacefulness and to believe that the worst is behind us.

I am saddened by the fact that this very country is systematically voting against releasing the intellectual property required to fabricate vaccines in other parts of the world, albeit temporarily. It's infuriating as hell. This crisis will not be over until this virus is eradicated from everywhere in the world.

And then there's the political gambling. China and Russia are donating vaccines only to countries who choose to avoid commenting on their (abysmal) human rights record. The USA is barely planning about exporting vaccines, instead of following Prof. Galloway's advice[1] to create a new "Marshall Plan" for the whole world.

The elites of the world are pure junk. But again, no surprises here, either. In the meanwhile, big pharma is grinning. Business as usual.

But not all is bad news. Today I am very proud of writing these lines

[1] https://www.profgalloway.com/calling-marshall/

on a Linux kernel, as I see Linus Torvalds[2] openly take position against stupidity and ignorance. And I am hopeful that the same technology that brought us the COVID vaccine could be used to fight AIDS[3], a pandemic that started exactly 40 years ago last Saturday[4].

[2] https://lkml.org/lkml/2021/6/10/957

[3] https://www.biopharma-reporter.com/Article/2021/04/15/Moderna-to-take-mRNA-flu-and-HIV-vaccines-into-Phase-1-trials-this-year

[4] https://www.cdc.gov/mmwr/preview/mmwrhtml/mm5021a1.htm

No Cookie Popup

Adrian Kosmaczewski

2021-06-18

After eight months of playing with this pure HTML website, the only thing I can say is: why didn't I do this sooner?

I am going to sound like a cranky old guy, but when I published my first web page 25 years ago (yup, August 1996, unbelievable) there weren't many choices to create "dynamic" websites, or as we called them back then, "databased web apps"[1].

(There actually was a magazine that lasted for a little while called "Databased Web", but I couldn't find any pictures of it online.)

As I was saying, to create a website with dynamic pages fed with a database, in 1996 you needed to use either Objective-C (yes, that one) with WebObjects; or you could use VBScript and SQL Server 6.5 if you were in the Microsoft galaxy; or you could use ColdFusion, which was really weird no matter how you looked at it. There wasn't even PHP yet!

Anyway. Going back to my first technology (pure HTML) has brought two great benefits. The first and most obvious one is speed. But the second benefit is a bit more subtle.

No more cookie popups. Have you noticed?

If you don't believe me, check your browser cookie settings, and look for cookies from the `https://akos.ma` domain.

You can't find any, because there isn't any. That is right. This website is naturally GDPR-compliant. Nothing here is tracking you. No ads, ~~no analytics~~, nothing.

The truth is, I don't want to gather analytics, because most websites doing that don't even look at the data they gather. And I know I wouldn't. So, why bother my readers with that?

If I really wanted to log visits to my site, I could do it on the server side, the good old way. But I don't care.

The only JavaScript I'm using here is for the search engine[2], it is small (~3.6 KB), and it is actually useful, because it brings you search re-

[1] https://www.amazon.com/Teach-Yourself-Active-Database-Programming/dp/1575211394

[2] /search

sults, and there's a lot of interesting stuff in this website for your reading pleasure. Which means that this page eats almost no power from your laptop battery.

In short, Hugo[3] is a triumph. It builds this whole website in about 2 seconds. It deploys in another 3 seconds. And it generates a fast website that looks good.

For the record, I created my first website in 1996 using Microsoft Word 6.0; yes, believe it or not. Microsoft, desperate to drown the threat of the web, released a plugin for Word that would export documents to HTML.

And yes, the resulting HTML was hideous. Like, FrontPage-kind of hideous.

Update, 2023-03-31: I've added analytics using TelemetryDeck App Analytics[4] today. They are 100% anonymous and do not require cookies of any kind.

Update, 2023-04-07 I've replaced TelemetryDeck with self-hosted Matomo[5] analytics, and I've set the instance to track without cookies[6]. The data is stored in the same server as this website you're visiting and is only accessible to the author.

[3] https://gohugo.io/
[4] https://telemetrydeck.com/
[5] https://matomo.org/
[6] https://matomo.org/faq/how-to/how-do-i-enforce-tracking-without-cookies/

Joplin

Adrian Kosmaczewski

2021-06-25

Note taking is very important to me. I keep everything in my notes, from ideas for blog posts like this one, to code snippets, to web pages, to plans of never started businesses, and so much more.

The Big Three

In my electronic note taking journey, I started, like many of us, using good old Evernote[1]. But around 2014 I got tired of the changes in the UI, and started looking elsewhere. I tried Microsoft OneNote[2] for a while; it is weirdly OK in many ways, yet it is also so clumsy in others, that I never totally clicked in it. Being stuck in the Apple galaxy for many years, I felt in the trap of moving to Apple Notes[3], a major mistake.

The biggest issue with the three big note taking apps I mentioned above is always the same: export. Once you use any of these apps, migrating to another is almost impossible. The worst offender is Apple Notes, by far, which also had the worse synchronization of all, because Apple has many talents, but iCloud synchronization isn't one of them. The app lost many of my notes, and many of the changes I made on my notes. Infuriating.

Bear and Others

I then used Bear[4]; I loved it and I was a paying customer of theirs for years. I truly believe that if you are working exclusively with Macs, iPads and iPhones, it is by far the best option. I know that Bear used the same synchronization engine than Apple Notes (iCloud) yet I have never had any problems with Bear. I guess they know how to use iCloud better than Apple. I stopped using Bear, though, I because I wanted a cross-platform solution for my notes.

[1] https://evernote.com/
[2] https://www.onenote.com/
[3] https://www.icloud.com/
[4] https://bear.app/

I also tried Simplenote[5], Org Mode for Emacs[6] (yes, I did try it), Boost Note[7], and nb[8]. As good as they are (for example, Simplenote had the fastest and greatest synchronization engine I've seen in ages) none of these worked for me, to be honest.

Enter Joplin

And then one day in January 2017 I discovered Joplin[9]. It was a relatively unknown project in GitHub[10] with a few thousand followers. I installed it and tried it.

And it was love at first use.

Joplin had already all the features I wanted:

- Cross-platform: Windows, Mac and Linux.
- Markdown-based.
- For plain-text notes and to-do lists.
- Alerts for to-do items.
- Notebooks, sub-notebooks, and tags.
- Searchable.
- CTRL+P for fuzzy-searching notes, notebooks, tags…
- Note history.
- Syncronization with Dropbox (and many other cloud services and mechanisms, including plain filesystem).
- Mobile app for iOS and Android.
- Integrated note encryption.
- Themable.
- Support for math content (equations, etc).
- Browser extension for Firefox & Chrome.
- Terminal application, with an `ncurses`-like TUI, and also with a nice set of CLI options (`joplin ls`, `joplin cat abcdef123`, etc).

These days, the GitHub project[11] has almost 24'000 followers. It has grown tremendously, it is fast, and I have literally left it open in the background of every computer I've used in the past 4.5 years.

The future of Joplin looks interesting; apparently they are working in their own sync server, which would be distributed as a Docker image, so that you can run it anywhere. I am sure I'm going to use it at some point.

Also, the desktop app features lots of nice new features: a spell checker; support for screenwriting with Fountain[12] and diagramming

[5] https://simplenote.com/
[6] https://orgmode.org/
[7] https://boostnote.io/
[8] https://xwmx.github.io/nb/
[9] https://joplinapp.org/
[10] https://github.com/laurent22/joplin
[11] https://github.com/laurent22/joplin
[12] https://fountain.io/

with Mermaid[13]; and sooo much more.

I’m grateful for this application!

[13] https://mermaid-js.github.io/mermaid/#/

More Rust Stuff

Adrian Kosmaczewski

2021-07-02

I keep learning about Rust and I have found really interesting bits all over the place. Here's a rough compilation.

Somebody figured out how to run Rust on iOS[1], on Android[2], and how to make macOS apps[3] with it. Because why not.

Here's a very interesting book: "Zero to Production in Rust"[4] about web services written in Rust. I wonder if Rust will end up displacing Go in this field. In any case, people are tracking whether are we web yet?[5] with Rust.

Of course, lots of people are interested in writing an OS in Rust[6], to which this introduction video for non-systems programmers[7] will be very interesting.

The rustlings[8] are useful exercises to read and write Rust code. It complements very well Rust by Example[9] and thankfully Julia Evans says[10] that Rust got easier to use since 2018.

Mathematicians are working on the RustBelt[11] project to provide mathematical foundation to Rust. And here's a paper[12] if you're into that kind of reading.

Rust is already supported[13] in a few serverless environments. Case in point, Rust in AWS Lambda[14].

---

[1] https://medium.com/visly/rust-on-ios-39f799b3c1dd

[2] https://medium.com/visly/rust-on-android-19f34a2fb43

[3] https://rymc.io/blog/2021/cacao-rs-macos-ios-rust/

[4] https://www.zero2prod.com/

[5] https://www.arewewebyet.org/

[6] https://os.phil-opp.com/

[7] https://www.youtube.com/watch?v=BBvcK_nXUEg

[8] https://github.com/rust-lang/rustlings

[9] https://doc.rust-lang.org/stable/rust-by-example/

[10] https://jvns.ca/blog/2018/01/13/rust-in-2018--way-easier-to-use/

[11] https://plv.mpi-sws.org/rustbelt/

[12] https://dl.acm.org/doi/pdf/10.1145/3158154

[13] https://dylananthony.com/posts/best-supported-serverless-languages

[14] https://dev.to/nicholaschiasson/beginner-s-guide-to-running-rust-on-aws-lambda-277n

Another interesting video, this time a talk at Stanford[15].

Some say that Rust is self driving C++[16], while others say that we should slow things down: Rust: “Move fast and break things” as a moral imperative[17]

And here’s another weird thing, at least in my opinion; using Rust to generate WebAssembly[18] and running it in the server with Deno.

Some frameworks and libraries I came across, that really picked my attention:

- Serde[19] for serialization and deserialization.
- Tokio[20] – asynchronous runtime for network apps.
- Krustlet[21] – Kubernetes Rust kubelet (experimental).
- ncurses[22] light wrapper.
- C2Rust[23] - a transpiler from C99 to Rust. It generates unsafe code mostly, but still, a very impressive tool. They’ve used to transpile Quake 3[24] and even a kernel module[25], learning a lot in the process.

And the winner is: Ruffle[26], a Flash player emulator. In Rust. Running on browsers as WebAssembly.

You heard right.

[15] https://www.youtube.com/watch?v=O5vzLKg7y-k
[16] https://blog.polybdenum.com/2017/07/22/rust-is-self-driving-c.html
[17] https://drewdevault.com/2021/02/09/Rust-move-fast-and-break-things.html
[18] https://thenewstack.io/using-web-assembly-written-in-rust-on-the-server-side/
[19] https://serde.rs/
[20] https://tokio.rs/
[21] https://krustlet.dev/
[22] https://crates.io/crates/ncurses
[23] https://c2rust.com/
[24] https://immunant.com/blog/2020/01/quake3/
[25] https://immunant.com/blog/2020/06/kernel_modules/
[26] https://ruffle.rs/

Testing LDAP

Adrian Kosmaczewski

2021-07-09

Testing applications that use LDAP for user authentication can be complicated. You just cannot use the production LDAP in your testing, because... reasons, so it can be difficult to make sure your application works properly before putting it in production.

Here go two simple tricks I've found to help me in those cases; I've used both and they work great.

Using Docker

This Docker container image[1] exposes an LDAP server in port 10389, ready to be used. Launch it with the common `docker run` command.

```
$ docker run --detach --rm --publish 10389:10389 rroemhild/test-openldap
```

Here's the configuration JSON you need for your typical Express application using Passport.js[2] for authentication:

```
const ldapConfig = {
  server: {
    url: 'ldap://localhost:10389',
    bindDN: 'cn=admin,dc=planetexpress,dc=com',
    bindCredentials: 'GoodNewsEveryone',
    searchBase: 'dc=planetexpress,dc=com',
    searchFilter: '(uid={{username}})'
  }
}
```

After connecting, just use these username/password combinations to authenticate in your application: `professor/professor`, `fry/fry`, `hermes/hermes` or `leela/leela`.

This can also work very well on a local Kubernetes cluster, of course, using Minikube[3] or K3d[4].

[1] https://github.com/rroemhild/docker-test-openldap
[2] https://www.passportjs.org/
[3] https://minikube.sigs.k8s.io/docs/
[4] https://k3d.io/

You can get even fancier exposing your service over an ngrok[5] TCP bridge to make it available over the Internet (requires a paid ngrok account, though, but it's totally worth it.)

Online

If for some reason you can't use Docker, here's an online LDAP server[6] ready to use for testing purposes.

The corresponding JSON configuration would be the following:

```
const ldapConfig = {
  server: {
    url: 'ldap://ldap.forumsys.com:389',
    bindDN: 'cn=read-only-admin,dc=example,dc=com',
    bindCredentials: 'password',
    searchBase: 'dc=example,dc=com',
    searchFilter: '(uid={{username}})'
  }
}
```

And now you can use names of scientists to connect: `euler/password`, `newton/password`, `tesla/password` and you're in.

[5] https://ngrok.com/

[6] https://www.forumsys.com/tutorials/integration-how-to/ldap/online-ldap-test-server/

Removing Singletons

Adrian Kosmaczewski

2021-07-16

Problem: your code has a big badass manager class with a singleton interface, and you would like to have more flexible, testable code.

The code samples in this article are in Swift, but the concepts translate to pretty much any modern language.

```
class Manager {
    static let singleton = Manager()
}
```

Recipe

Create a protocol called `ManagerProtocol` (Objective-C, Swift) or an interface `ManagerInterface` (Kotlin, Java, C#), or an abstract class with pure virtual methods `IManager` (C++) to contain the signatures of the public properties of your manager.

```
protocol ManagerProtocol {}
```

Make the singleton return a `ManagerProtocol`, instead of a `Manager`.Make your class comply to the protocol; that is, implement the interface.

```
class Manager: ManagerProtocol {
    static let singleton: ManagerProtocol = Manager()
}
```

Now your code will most probably not compile; add the signatures of the public methods and properties that you client code requires to the protocol / interface until everything compiles again.

In the classes that use the singleton, create a public property of type `ManagerProtocol` and initialize it to the value of the singleton at initialization or construction time: in other words, turn this

```
class Client {
    func method() {
        Manager.singleton.doThat()
    }
}
```

into this

```
class Client {
    var manager: ManagerProtocol = Manager.singleton

    func method() {
        self.manager.doThat()
    }
}
```

Replace every instance of the singleton call with a call to the ivar.

Result

Now your components are decoupled; you can replace the Manager class at runtime with another object, and this is useful for testing your code. It can also be useful at runtime, to switch the behaviour of your code from one implementation to another if needed.

Another side effect is that you have a nice interface / protocol that describes your “manager” as an abstract entity, and this is a powerful tool for documenting the code and establishing relationships among things.

Password Hashing in Django

Adrian Kosmaczewski

2021-07-23

This technique can be useful when migrating applications from Django to ASP.NET or PHP, keeping usernames and passwords intact.

Passwords in Django are stored in two ways (in the `core_vipassuser` table of the database):

1. When accounts have been created without a valid password, or using Facebook or any other OAuth2 provider, the passwords are stored like this: `!mafzMhEywOfhrFMvFJ16JhB1uQiAvHRaN4KuEqfg` which is a random string prefixed with `!`. This is shown in the `make_password()` function in django.contrib.auth.hashers[1] when the `password` parameter is `None`. This prevents usage of the system with an empty password.

```
UNUSABLE_PASSWORD_PREFIX = '!'
UNUSABLE_PASSWORD_SUFFIX_LENGTH = 40

def is_password_usable(encoded):
    """
    Return True if this password wasn't generated by
    User.set_unusable_password(), i.e. make_password(None).
    """
    return encoded is None or not encoded.startswith(UNUSABLE_PASSWORD_PREFIX)

def make_password(password, salt=None, hasher='default'):
    if password is None:
        return UNUSABLE_PASSWORD_PREFIX + get_random_string(UNUSABLE_PASSWORD_S
```

OAuth2 tokens are stored in the `social_auth_usersocialauth` table.

2. Django passwords are hashed using PBKDF2[2] with test vectors here[3]. There are implementations of this algorithm in several languages. Hashed passwords in a Django DB look like this: `pbkdf2_sha256$36000$5LjfzfBwQAVI$sbEcyHm7a27GFKOgOOymu+mauqVLhS2QKQE4yLk8B9` where the following elements are separated by `$`:

    1. `pbkdf2_sha256` indicates the hashing algorithm

[1] https://docs.djangoproject.com/en/2.1/_modules/django/contrib/auth/hashers/
[2] https://www.ietf.org/rfc/rfc2898.txt
[3] https://www.ietf.org/rfc/rfc6070.txt

2. `36000` is the number of iterations
3. `5LjfzfBwQAVI` is the salt
4. `sbEcyHm7a27GFKOgOOymu+mauqVLhS2` is the actual hash.

This format is referred as `<algorithm>$<iterations>$<salt>$<hash>` in the Django documentation:

> By default, Django uses the PBKDF2 algorithm with a SHA256 hash, a password stretching mechanism recommended by NIST. This should be sufficient for most users: it's quite secure, requiring massive amounts of computing time to break.

Validating Django Users from PHP

Using that information, the following code takes a password (given by the user) and creates the same hash as in point 2.4 above; that can be used to validate that the user is effectively whoever they claim to be.

```php
<?php
hash_pbkdf2('sha256', 'password', 'salt', 36000, 32, true);
```

The `hash_pbkdf2()` function is available since PHP 5.5.

Validating Django Users from C

```
using System;
using System.Text;
using System.Security.Cryptography;

namespace App
{
    class Program
    {
        static string PBKDF2(string password, string salt, int iterations, int
        {
            byte[] saltBytes = Encoding.UTF8.GetBytes(salt);
            using (Rfc2898DeriveBytes pbkdf2 = new Rfc2898DeriveBytes(password,
            {
                byte[] resultBytes = pbkdf2.GetBytes(size);
                return Convert.ToBase64String(resultBytes);
            }
        }

        static void Main(string[] args)
        {
            Console.WriteLine(PBKDF2("password", "salt", 36000, 32));
        }
    }
}
```

More information

Password management in Django[4], and GitHub - defuse/password-hashing: Password hashing code.[5], where the code above was adapted from.

[4] https://docs.djangoproject.com/en/2.1/topics/auth/passwords/
[5] https://github.com/defuse/password-hashing

Video Editing in Linux

Adrian Kosmaczewski

2021-07-30

Let's talk about the worst part of my Linux experience[1] so far: video editing. It's not like I'm editing videos every day in my Linux machines, but every so often I have to do something around video, and seriously, it's a PITA.

The biggest pain point in the Linux platform is, without any doubt, non-linear video editors[2]. I explored many options while creating the short-lived VSHN.timer[3] video series from October 2020 to March 2021. Those videos were short; 3 minutes tops, so the projects were never very complicated or required lots of resources. Yet I had enough headaches with each one of them, as to drop the idea of doing them altogether.

And when I mean headaches, I mean: crashes; unusable user interfaces; audio input and output trouble; GPU support (or not); and more crashes. Oh, did I mention crashes? Yeah.

Frankly, and unsurprisingly, the best part of editing video in Linux is… using FFmpeg[4] on the command line. That's it; the rest goes downhill from here. Modifying video files on the terminal is surprisingly straightforward and predictable: extract clips, change formats, all of that works extremely well. Non-linear editors? Nah.

TL;DR: KDEnlive[5] FTW.

Shotcut

I used Shotcut[6] for the first few editions of VSHN.timer; and basically, it crashed all the time. I barely could finish the one or two videos I wanted to edit with it.

[1] /blog/migrating-from-macos-to-linux/
[2] https://en.wikipedia.org/wiki/Non-linear_editing
[3] /blog/vshn.timer/
[4] https://ffmpeg.org/
[5] https://kdenlive.org/
[6] https://www.shotcut.org/

Flowblade

I used Flowblade[7] for a few more VSHN.timer editions, and although it didn't crash or anything, I found the UI quite unusable. In particular the controls are extremely small, and it's all too easy to click on the wrong places. Disappointing.

OpenShot

OpenShot[8] was much more stable than Shotcut, and it's easier to use. The brilliant thing about it is the story[9] of the developer behind it: a C#/.NET developer without experience in Linux started the project in May 2008… and here it is now.

The project includes a whole C++ library for video and audio processing, but all in all, I didn't like it, I think it was not very usable. The UI was quite confusing, and it kept losing audio input randomly here and there.

DaVinci Resolve

DaVinci Resolve[10] is apparently one of the biggest players in the market, used for big movies and everything.

In my experience, the biggest problem with DaVinci Resolve is that none of the two Linux machines I'm using (a Lenovo ThinkPad Carbon X1 with 16 GB RAM for work, and my personal TUXEDO InfinityBook Pro 14 v4 with 32 GB RAM) has a GPU supported by it, and as such, I could not make it work in neither machine. It installs, but does not start. It is unclear in the documentation which machines are supported and which aren't.

Infuriating; why isn't this detected at installation time?

Lightworks

Lightworks[11] is another "big player" in the market. It looks very polished, very professional and filled with options and filters. When I first tried it in 2020, it didn't work with the audio on my ThinkPad, and the UI was extremely unstable; movie clips were disappearing from the UI and such. Unusable.

I tried it again a few months later in 2021, and although it seemed a bit more stable than previously, it was impossible to find out how to select the audio input for voiceover (I finally found out how to do it in

[7] http://jliljebl.github.io/flowblade/
[8] http://www.openshot.org/
[9] https://www.openshot.org/story/
[10] https://www.blackmagicdesign.com/products/davinciresolve/
[11] https://www.lwks.com/

the settings panel of the application) but I found it very cumbersome to add transitions, the app kept throwing unintelligible errors all over the place.

Finally, Lightworks being a commercial application, one needs a “pro” account to export to 1080p or higher. I didn’t find it worth the money.

Lumiera

I discovered Lumiera[12] last month, and apparently it’s a rewrite of another video editor called Cinelerra-CV[13]; never heard of either. I haven’t tried neither of these yet, so I can’t comment. I’m just adding it here for reference.

Pitivi

Another one I haven’t tried yet, but appears here for reference: Pitivi[14].

KDEnlive

After all the back and forth, KDEnlive[15] is the editor I used for most VSHN.timer videos. It has a very nice effect editor with keyframes (much more usable than in Flowblade) and in general, it has a very stable and snappy interface. It allowed me to work smoothly, simply, and with almost no crashes at all.

It isn’t available for Mac though[16]; only for Windows & Linux. As usable as it is, I struggled a bit to find out how to add a fade in[18] effect.

All things considered, KDEnlive is the one I kept using the longest, and the one I recommend to other Linux users for their video editing needs.

[12] https://lumiera.org/

[13] http://cinelerra-cv.wikidot.com/

[14] https://www.pitivi.org/

[15] https://kdenlive.org/

[16] **Update, 2023-04-28:** There is now (since October 2021, apparently[17]) a Kdenlive build for macOS.

[18] https://kdenlive.org/en/project/howto-fading-inout-kdenlive-titles/

Alpine Linux in VirtualBox

Adrian Kosmaczewski

2021-08-06

I've been playing with Alpine Linux[1] on VirtualBox[2], and here are some notes I took during the process.

Installation

Let's get started:

- Create a NAT Network in the VirtualBox preferences, call it "MyNetwork".
  - More information here[3].
- Download[4] the latest x86_64 version in ISO format.
- Create a virtual machine in VirtualBox.
  - Name it "Alpine Linux".
  - Attach the ISO image to the virtual machine.
  - Select "Attached to NAT Network" as adapter, and select the "MyNetwork" option if not selected.
  - Choose "Bridged Adapter" if you want to access the VMs from the host and vice-versa.
- Boot and run the `setup-alpine` command to get started. Reply all questions and wait for system to be installed.
  - Choose a proper hostname, so that another machine has a different hostname in the prompt.
- Shutdown with the `poweroff` command, remove the ISO from the virtual disk drive.
- Start the VM, it should work.

Applications

- Run `vi /etc/apk/repositories` and uncomment both lines with `community` repositories.
- Run `apk update`.
- Install some common stuff: `apk add sudo vim emacs python3 mplayer bash zsh tmux git curl wget bat fortune jq tig mc mandoc man-pages`

[1] https://alpinelinux.org/
[2] https://www.virtualbox.org/
[3] https://www.nakivo.com/blog/virtualbox-network-setting-guide/
[4] https://alpinelinux.org/downloads/

Pandoc

For Pandoc[5] we need to install Haskell and a few more things first:

```
# apk add cabal ghc libc-dev zlib-dev
# cabal update
# cabal install pandoc
```

Create Users

You don't want to be root all the time:

- `adduser username`
- Follow the steps here[6] to install build tools and add the new user to have `sudo` powers:
  - `visudo`
  - Add the line `username ALL=(ALL) ALL` below the similar one for `root`
  - `apk add alpine-sdk`
- Exit root and login as new username:
  - `sudo apk add htop` should work

Shell

My favourite shell is zsh[7]:

- `apk add shadow` (there's no `chsh` in Alpine)
- `chsh -s $(which zsh)`
- `sh -c "$(curl -fsSL https://raw.github.com/ohmyzsh/ohmyzsh/master/tools/inst`
  to install Oh-My-Zsh[8]

Connect to Another VM

Let's repeat the process, so that we can `ssh` from one VM to another:

- Create a second VM with the same procedure as above, and create the same username.
- Run `ifconfig` in both to find the IP addresses.
- `ssh XX.XX.XX.XX` (that is, connect from one to the other.)
- Launch `tmux` in one VM and `tmux att` in the other.
- Type and see how things happen in both at the same time.

[5] https://pandoc.org/

[6] https://wiki.alpinelinux.org/wiki/Include:Setup_your_system_and_account_for_building_packages

[7] https://www.zsh.org/

[8] https://ohmyz.sh/

Xfce

Xfce is a nice and snappy desktop environment. The install instructions[9] are very straightforward:

```
# setup-xorg-base
# apk add xfce4 xfce4-terminal lightdm-gtk-greeter xfce4-screensaver dbus-x11
# apk add virtualbox-guest-additions virtualbox-guest-modules-virt
# rc-service dbus start
# rc-service lightdm start
```

By default one can't shutdown or reboot from the Xfce UI, though; easy fix:

- `apk add polkit consolekit2`

Networking from ISO Image

When booting from the ISO image, all changes are lost!

Write this into `/etc/network/interfaces`

```
auto lo
iface lo inet loopback

auto eth0
iface eth0 inet dhcp
```

Then type the command `/etc/init.d/networking restart` and you should be able to `ping google.com` once again.

Cloning VMs

When cloning VMs in VirtualBox, pay attention to the fact that it also clones the MAC address of the interface… hence the clone shares the same IP address as the original VM! Don't forget about this.

Stuff that doesn't (can't) work on Alpine

Some software I tried to install actually requires glibc[10], but since Alpine uses musl[11], it can't run:

- Emacs GUI
- Kdenlive[12]

[9] https://wiki.alpinelinux.org/wiki/Xfce_Setup
[10] https://www.gnu.org/software/libc/
[11] https://musl.libc.org/
[12] /blog/video-editing-in-linux/

Fibonacci

Adrian Kosmaczewski

2021-08-13

I have a bit of a trip with Fibonacci numbers lately; here are many things I've learnt about them lately.

Formula

There this thing called Binet's formula[1], which allows us to calculate any Fibonacci number.

First we need to calculate these terms:

$$a_n = \left(\frac{1 + \sqrt{5}}{2} \right)^n = \phi^n$$

$$b_n = \left(\frac{1 - \sqrt{5}}{2} \right)^n = -(\phi - 1)^n$$

where ϕ is the solution to $x^2 - x - 1 = 0$ which gives Binet's formula:

$$f_n = \frac{a_n - b_n}{\sqrt{5}}$$

The a term grows substantially faster than b; the contribution of b is negligible after `n ~ 10`.

n	a(n)	b(n)	diff %	a / sqrt(5)	(a - b) / sqrt(5)
0	1	1	0	0.45	0
1	1.62	-0.62	61.80%	0.72	1
2	2.62	0.38	85.41%	1.17	1
3	4.24	-0.24	94.43%	1.89	2
4	6.85	0.15	97.87%	3.07	3
5	11.09	-0.09	99.19%	4.96	5
6	17.94	0.06	99.69%	8.02	8

[1] https://en.wikipedia.org/wiki/Jacques_Philippe_Marie_Binet

n	a(n)	b(n)	diff %	a / sqrt(5)	(a - b) / sqrt(5)
7	29.03	-0.03	99.88%	12.98	13
8	46.98	0.02	99.95%	21.01	21

About 1/89

The number $\frac{1}{89}$ is weirdly related[2] to the Fibonacci series.

$$\frac{1}{89} = \sum_{n=1}^{\infty} F_n 10^{-(n+1)}$$

where F_n is the nth Fibonacci number.

```
  .01
  .001
  .0002
  .00003
  .000005
  .0000008
  .00000013
  .000000021
  .0000000034
  .00000000055
  .000000000089
  .0000000000144
        .
        .
+       .
----------------
  .01123595505... = 1/89
```

Magazine

There is an entire mathematical magazine[3] dedicated to the Fibonacci series. It has existed since 1963, and is still being published.

Python

This is one of the weirdest algorithms I've come across to calculate Python, with lots of bitwise operators.

```
def fib(n):
  return (4 << n*(3+n)) // ((4 << 2*n) - (2 << n) - 1) & ((2 << n) - 1)
```

(Source)[4]

[2] https://www2.math.ou.edu/~dmccullough/teaching/miscellanea/miner.html
[3] https://fq.math.ca/
[4] https://blog.paulhankin.net/fibonacci/

I had to look up about them, it's not like I use Python bitwise operators[5] every day:

```
x << y == x * (2 ** y)
```

Ergo, the code above is roughly the same as:

```
def fib(n):
  a = (4 * (2 ** (n*(3+n))))
  b = ((4 * (2 ** (2*n))) - (2 * (2 ** n)) - 1)
  c = ((2 * (2 ** n)) - 1)
  return a // b & c
```

The & operation is the bitwise AND[6] operation, which, it turns out, is equivalent to the MODULO operation when dealing with powers of two[7].

> In general, if divisor is a power n of two, the modulo operation can be translated to a bitwise AND with divisor-1. Similarly, the integer division value / divisor corresponds to a right shift of n positions: value » n.

(Source)[8]

1000th Fibonacci number

And to close this off, this is the 1000th one:

```
26863810024485359386146727202142923967616609318986952340123175997617981700247881689338369654483356564191827856161443356312976673642210350324634850410377680367334151172899169723197082763985615764450078474174626
```

All in all, 209 characters long.

[5] https://wiki.python.org/moin/BitwiseOperators
[6] https://en.wikipedia.org/wiki/Bitwise_operation#AND
[7] https://stackoverflow.com/a/3072710
[8] https://mziccard.me/2015/05/08/modulo-and-division-vs-bitwise-operations/

Computers Bundled With Linux

Adrian Kosmaczewski

2021-08-20

For those interested in buying a computer with Linux built-in, here's a list of options. There might be more, but these are the ones I found by scouting around.

Based in Europe

These manufacturers are based on Europe:

- Tuxedo Computers[1] (Germany)
  - VAT discounted when buying from Switzerland.
  - Lots of customization options.
  - I'm using a Tuxedo laptop to write this post.
- Slimbook[2] (Spain)
- List[3] of manufacturers bundling ZorinOS:
  - StarLabs[4] (UK)
  - Laptop With Linux[5] (Netherlands)
  - SKIKK[6] (Netherlands)

USA

These are located in the states.

- System76[7]
  - Essentially, many of the laptops sold by System76 are the same ones offered by Tuxedo.
  - They have their own flavor of Linux: Pop!_OS[8].
- Purism[9]
- There used to be Zareason[10] too, but they closed last year due to the pandemic.

[1] https://www.tuxedocomputers.com/
[2] https://slimbook.es/
[3] https://zorinos.com/computers/
[4] https://starlabs.systems/
[5] https://laptopwithlinux.com/
[6] https://www.skikk.eu/
[7] https://system76.com
[8] https://pop.system76.com/
[9] https://puri.sm/
[10] https://zareason.com/

Big Brands

These days you can get pre-installed Linux from some big manufacturers in Switzerland, which means you get to avoid the “Windows tax” on your hardware, which amounts to roughly 150 Swiss francs. For that money, you can get more storage or RAM. Finally.

- Dell XPS 13’’[11]
- Lenovo X1 Carbon[12]
  - Can be bought without any operating system, with a discounted price.
  - I’m using one of these at work, and I know my next laptop will be a Lenovo X1 Carbon for sure.

Others

There is also the Pinebook Pro[13] which can run various Linux and BSD operating systems.

Update, 2022-03-22: There’s the Framework Laptop[14] now as well.

Update, 2022-10-28: Minifree[15] offers GNU+Linux laptops with coreboot, osboot or libreboot preinstalled.

[11] http://www.dell.com/ch/p/xps-laptops?c=ch&cs=chbsdt1&l=fr&s=bsd&~ck=mn
[12] https://www.lenovo.com/ch/en/laptops/thinkpad/thinkpad-x1/c/thinkpadx1
[13] https://pine64.com/product-category/pinebook-pro/
[14] https://frame.work/
[15] https://minifree.org/

Radio Silence

Adrian Kosmaczewski

2021-08-27

I have had three episodes of burnout in my professional life, in three consecutive years. Not the best moments, for sure. This story is about the first episode, less than a decade ago.

I had been working at the company I was at the time for already almost a year, when I started having strong episodes of depression and anxiety.

At that moment I was working in a both complex and complicated project that was (as I would learn later) literally defying some laws of physics, and thus could not be done, at least not in this universe. But nobody knew at that moment, or dared to tell me. And, even worse, nobody was telling this to the customer, either, even though we were "agile" and all of that. I actually thought somebody was communicating our project issues to them, as I was not in direct contact with them. More on that later.

I started having panic attacks fairly often, particularly at night, and I could not sleep well. The previous year I had gone through a deep personal crisis.

When I realized I was having a big problem, I chose to do the professional thing: I sent an e-mail to my boss and to the owner of the company where I was working, explaining the situation and that I needed to talk to them in person.

To this day, **I still haven't received a reply to that e-mail**. Not even an out-of-office message. Nothing. Radio silence.

What's worse, is that I know they received it, and read it. They chose not to reply. I don't know why, and probably never will.

A few days later I had to meet the customer of this project by myself (they were in another city). I told them what I thought they knew, which was that defying laws of physics is generally a bad idea. They were dumbfounded, as they thought that the project was almost ready to ship.

Or, at least, that's what our product owner was telling them.

Ready. To. Ship. We hadn't even been able to run a single prototype.

I fell in a strong depression that night. I was alone in my hotel room, somewhere else than home, and I could not believe that this was happening.

And I sank.

Of course I blamed myself. Because that's what happens in my head when I have depression. I thought this was all my failure. I have failed this project. I was failing at work, hence I was failing to perform my duty in life. I was failing as a person. It all dawned on me, as a dark black night in the woods without a flashlight.

Long story short, to this day, I have never received any reply from my "boss" or anyone else. A few days later I went into medical leave, and a few months later I left the company. Nobody came to say goodbye from the team. I just closed the door and left.

I had two more episodes of burnout in the following years. In the second one I ended up taking medication, and in the meantime, went to therapy for two years. It took me four years to finally feel like I could go back to a normal life.

My wife supported me like nobody else, with infinite patience and love. I don't have words to express my love for her. I owe her everything.

Regarding this issue, or other issues in various shitty workplaces I have had the misfortune of working at, I do not blame myself for anything else now, other than for having chosen to work in those.

Because I deserved support in that moment. I deserved having my boss taking me to the side to have a coffee and asking me how I was doing. I deserved having somebody more in my team to help me figure out a new technical solution for that project, instead of leaving me alone. I deserved having a product owner communicating issues to that customer. I deserved having someone giving a fucking shit about my mental health.

I deserved a reply to that e-mail I sent; I did not deserve to have that radio silence instead.

Linux Package Manager Strategy

Adrian Kosmaczewski

2021-09-03

There are quite a few package managers available in Linux, and it's hard to figure out which one to use for what. What I'm going to recommend here is my personal take after lots of back and forth, and trying new things.

Also, please bear in mind that I'm an Ubuntu user, and of course, other distributions have other possibilities, requirements, and best practices.

TL;DR:

In general I apply the following algorithm when looking for software to install in my laptop:

1. If the app is a GUI app, use Flatpak.
2. If the app is a command-line tool, use APT.
3. If the tool is programming-language specific, just use the package manager of that language (`pip install`, `gem install`, `npm install`, etc)
4. If the tool can be installed by `curl | sh` some script, first read the script, the use it to install it.
5. If all else fails, download the AppImage.
6. By all means, don't use Snap.
7. Forget about Homebrew.

Flatpak

I've started using Flatpak[1] recently, and it works great for GUI applications. Most of the desktop apps I'm using (I'd say 95% of them) are installed with it. If you are looking for a simple way to get some piece of software on your Linux box, I'm sure it is already available in Flathub[2].

Another thing that I like about Flatpak is that it works in many different distributions.

[1] https://flatpak.org/

[2] https://flathub.org/

The only app I did not install with Flatpak is Visual Studio Code, which is severely crippled when installed through Flathub. I did not find Shutter in it, either.

I also tried to install Microsoft Teams for Linux (don't ask) but it simply does not work. Of course this is Microsoft's fault, not Flatpak's; so I just login with Chrome when I need it (which thankfully it's not very often.)

Advanced Package Tool

The default APT[3] (`apt-get install`) package manager is by far the best option to get software on your box.

Of course we all know that most of the packages available there are usually a bit outdated, but the truth is that you can `add-apt-repository` and get lots of help from the community.

Take for example the excellent Shutter[4] screenshot tool; somebody came up[5] with a suitable repository to get that great app back on your system.

Another tool that I installed with APT is Visual Studio Code, for which Flatpak is not a good option. I did the same for Emacs 27[6].

Other Options

There's a lot more options available in Ubuntu to get software.

AppImage

Finally, there are plenty of great apps available using the AppImage[7] format. In the rare cases where I cannot get an app through Flatpak or APT, I just download the AppImage to my `~/.local/bin` folder, make it executable (`chmod +x`), and we're done.

Update, 2022-03-22: If you use AppImage, try AppImageLauncher[8], a "helper application for Linux distributions serving as a kind of"entry point" for running and integrating AppImages."

[3] https://en.wikipedia.org/wiki/APT_(software)

[4] https://github.com/shutter-project/shutter

[5] https://ubuntuhandbook.org/index.php/2021/08/nstall-shutter-ubuntu-20-04-official-ppa/

[6] https://launchpad.net/~kelleyk/+archive/ubuntu/emacs

[7] https://appimage.org/

[8] https://github.com/TheAssassin/AppImageLauncher

Snap

Ubuntu users have this thing called Snap[9] (`snap install`) to add applications. In general I have disliked using it, and this is mostly because of the sandbox thing. Applications are crippled in many subtle ways, and I stopped using it altogether.

Homebrew

Yes, Homebrew[10] also works in Linux, but no, I don't use it at all.

[9] https://snapcraft.io/
[10] https://brew.sh/

How to Write a Programming Book

Adrian Kosmaczewski

2021-09-10

Writing a programming book is not very complicated, to be honest: it just consists of putting one word after another. Shocker! I'm not kidding.

I assume that if you're reading this, you are interested in writing a book. In that case, the best you can do to write your book is to have a deadline, and then keep a simple routine to achieve it: write at least 500 words a day. That's it.

Of course, it's easier said than done, at least at the beginning. When I began writing, I could barely reach the 500 words mark after hours of trying. Now it only takes me twenty minutes to get that far, and sometimes I can get at least a thousand relatively coherent words per hour, sometimes even more. As Joel Spolsky said, writing is a muscle, and the more you do it, the easier it is.

But how much is a thousand words? To have an idea, a book of 250 pages (without code or illustrations, just text) is approximately a volume of 50'000 words. Books with code snippets have less words, maybe 30'000, but no less information; actually more, because of the higher density of information in said code snippets. My second book[1] was around 30'000 words, precisely, around 280 pages. Which means that, at a rate of 500 words a day, I needed 60 days of work (around 12 weeks, at 5 days each) to finish… the draft.

Because this is very important: do not edit during the draft writing phase. Never. Just… write. Pour text on the screen. Do not do anything else than writing. If you start editing yourself… well, you'll never finish the book. Don't censor yourself during that writing phase. Get the draft out.

But wait, what about inspiration? It doesn't matter, really. Write even if you're not inspired, even if it hurts, even if you hate the words that come out. Write. Get your 500 words even if you know that it's utter garbage. Get. The. Draft. Out.

Maybe it's not utter garbage, but you're in a bad day. Maybe you'll find something great in those ugly words in the future. For the moment, just write. I know, it sucks. But it works. Believe me.

[1] https://www.oreilly.com/library/view/sencha-touch-2/9781449339371/

After the draft comes edition (that is, the self-censorship part). I personally prefer, for this phase, to print the text on good old paper, go to the nearest café, and read it, away from everything, just me, the draft, some coffee, and a highlighter marker. A yellow one. Never green or red. I'm kidding, any color will do. OK, yellow is better.

Do not edit your draft immediately, though; wait a whole week before starting the editing phase. It is important to "forget" what you wrote a little bit before editing it.

But what would be a programming book without code snippets? Well, the rule of thumb is the following: first write the code, then write the text that explains the code. That's it. Many of the most important books in programming have been written this way. Dave Mark[2] told me this, and he knows a thing or two about writing programming books.

I know, I should have stated this last rule first. Sorry about that. I hope you haven't written many pages of your draft before reaching this paragraph. First the code, then the text.

How does edition work? You will read, edit, read, edit, and re-read, and re-edit your draft a gazillion times. Not really that much, but a lot, or until it reads well.

But what does "reads well" mean? You'll know when you read yourself and you like what you wrote. There is no other gauge. If you like what you read, then it's done.

And when will you be done with your book? Never, probably, but that's why publishing exists: as Jorge Luis Borges[3] once said, publishing is a great way to stop editing. Generate a PDF, sell it on Gumroad[4], Leanpub[5] or Lulu[6], and you're done.

(Don't sell your book in Amazon, please.)

With these simple rules you should get your book project done in a few months.

Another thing: don't use Word[7], LibreOffice[8], Pages[9], AbiWord[10] or Calligra Words[11] or anything like that. Those editors mess your mind with toolbars filled with options and buttons and menus and things that will distract you from the important bit: the text itself.

Because what's important is to Get. The. Draft. Out. Capisce?

[2] https://twitter.com/davemark
[3] https://en.wikipedia.org/wiki/Jorge_Luis_Borges
[4] https://gumroad.com/
[5] https://leanpub.com/
[6] https://www.lulu.com/
[7] https://en.wikipedia.org/wiki/Microsoft_Word
[8] https://www.libreoffice.org/
[9] https://www.apple.com/pages/
[10] https://www.abisource.com/
[11] https://calligra.org/

Instead, use the closest plain text editor you can get your hands on: Notepad[12], Notepad++[13], Vim[14], Emacs[15], Visual Studio Code[16], TextEdit[17], EditPlus[18], WordGrinder[19], Gedit[20], ~~TextWrangler~~[21], BBEdit[22], Sublime Text[23], TextMate[24], Ulysses[25], Apostrophe[26], UltraEdit[27], Joplin[28], etc. I've written extensively about text editors (and keyboards) in one of my books[29], so I'll just let you download it and read it if you want to know more. It's a funny book, you should read it.

Remember to create separate files for each chapter, and of course, use version control; Git[30] or something else that you feel confortable with. Yes, Subversion[31] is fine, if that's your thing. Version control is like undo on steroids. You can also open an account on GitHub[32] or GitLab[33] and get a backup copy of your whole project on a private repository there. Backups are important. Backup your book.

(If version control is too geeky for you, well, maybe writing a programming book would be already a challenge in itself; but if that's the case, Dropbox[34] has some kind of version management thing integrated[35], but somewhat restricted for free accounts. I prefer Git + GitLab in any case.)

With a plain text editor you can of course save your work as a bunch of `.txt` files, but I personally use Asciidoc[36]. It is plain text with some markers for titles, bold text, and more. There is also the venerable and über popular Markdown[37] format, which is similar, but I prefer Asciidoc instead of Markdown for books.

[12] https://en.wikipedia.org/wiki/Windows_Notepad
[13] https://notepad-plus-plus.org/
[14] https://www.vim.org/
[15] https://www.gnu.org/software/emacs/
[16] https://code.visualstudio.com/
[17] https://en.wikipedia.org/wiki/TextEdit
[18] https://editplus.com/
[19] https://cowlark.com/wordgrinder/
[20] https://en.wikipedia.org/wiki/Gedit
[21] https://www.barebones.com/products/textwrangler/
[22] https://www.barebones.com/products/bbedit/
[23] https://www.sublimetext.com/
[24] https://macromates.com/
[25] https://ulysses.app/
[26] https://gitlab.gnome.org/World/apostrophe
[27] https://www.ultraedit.com/
[28] https://joplinapp.org/
[29] /books/Tales_Of_Editors_And_Keyboards/Tales_Of_Editors_And_Keyboards.pdf
[30] https://git-scm.com/
[31] https://subversion.apache.org/
[32] https://github.com/
[33] https://gitlab.com/
[34] https://www.dropbox.com/
[35] https://www.dropbox.com/features/cloud-storage/file-recovery-and-history
[36] https://asciidoctor.org/
[37] https://daringfireball.net/projects/markdown/

I think Markdown is great for blog posts (like this one), and shorter texts; but nothing beats Asciidoc for longer works, like books, precisely. Of course, if you like LaTeX[38], that's another great option.

If you want a longer version of what you just read, from a real rock star in the world of programming book writing, and with lots of great information about publishing houses and royalties and more, check out Scott Meyers' (of "Effective C++" fame) own guide[39] to writing technical books.

In short, writing consists of a set of very simple rules applied iteratively. It sounds awfully similar to functional programming. And both are equally wonderful.

Update, 2022-11-27: Following a very interesting comment on Mastodon[40] I would like to point out that this process does not imply a full picture of the first draft, and I hope that this text does not convey such an idea. As a matter of fact, it's quite the opposite situation: **writing is thinking**. As the words flow through your fingers, they'll be processed by your brain. Don't censor yourself. Let your words flow. They will model your thoughts as they do.

[38] https://www.latex-project.org/

[39] https://www.aristeia.com/authorAdvice.html

[40] https://oc.todon.fr/@famuvie/109415024270300522

Amazon EKS Anywhere

Adrian Kosmaczewski

2021-09-18

I've been playing lately with Amazon EKS Anywhere[1], the Kubernetes distribution used by EKS[2] that you can install in your own premises. It works pretty well and makes for a very decent alternative to Minikube[3], K3s[4] and other similar "Kubernetes-in-your-laptop" packages.

Here are the steps I followed in my Ubuntu box:

Install tools

Very simple to install, just `curl` a few tools and you're done.

```
$ curl "https://github.com/weaveworks/eksctl/releases/latest/download/eksctl_$(
$ mv /tmp/eksctl ~/.local/bin
$ export EKSA_RELEASE="0.5.0" OS="$(uname -s | tr A-Z a-z)"
$ curl "https://anywhere-assets.eks.amazonaws.com/releases/eks-a/1/artifacts/ek
$ mv ./eksctl-anywhere ~/.local/bin
```

Create cluster

Once you have installed it, very easy to create a new cluster:

```
$ eksctl anywhere generate clusterconfig dev --provider docker > dev.yaml
$ eksctl anywhere create cluster -f dev.yaml
```

Deploy app

And of course, you can launch an app to see it in action:

```
$ kubectl apply --kubeconfig ./dev/dev-eks-a-cluster.kubeconfig -f "https://any
$ kubectl port-forward svc/hello-eks-a 3000:80 --kubeconfig ./dev/dev-eks-a-clu
$ curl localhost:3000
$ k9s --kubeconfig dev/dev-eks-a-cluster.kubeconfig
```

[1] https://aws.amazon.com/eks/eks-anywhere/
[2] https://aws.amazon.com/eks/
[3] https://minikube.sigs.k8s.io/docs/
[4] https://k3s.io/

Delete Cluster

Finally, some cleanup:

```
$ eksctl anywhere delete cluster dev
```

Vagrant, k3s and VirtualBox

Adrian Kosmaczewski

2021-09-23

Last weekend I decided to learn Vagrant[1] to build a simple k3s[2] Kubernetes cluster on top of a set of VirtualBox[3] virtual machines.

The result of those explorations is now available on my GitLab[4] for you to use and enjoy.

The `Vagrantfile` simply launches 4 VMs based on Alpine 3.14; one for the k3s "server" (the control plane) and three agents (aka worker nodes). The `kubeconfig` file is copied to a shared folder, so that one can easily `kubectl` or even better k9s[5] on the new cluster, to install applications or do whatever you want.

```
# -*- mode: ruby -*-
# vi: set ft=ruby :

server_ip = "192.168.33.10"

agents = { "agent1" => "192.168.33.11",
           "agent2" => "192.168.33.12",
           "agent3" => "192.168.33.13" }

# Extra parameters in INSTALL_K3S_EXEC variable because of
# K3s picking up the wrong interface when starting server and agent
# https://github.com/alexellis/k3sup/issues/306

server_script = <<-SHELL
    sudo -i
    apk add curl
    export INSTALL_K3S_EXEC="--bind-address=#{server_ip} --node-external-ip=#{s
    curl -sfL https://get.k3s.io | sh -
    echo "Sleeping for 5 seconds to wait for k3s to start"
    sleep 5
    cp /var/lib/rancher/k3s/server/token /vagrant_shared
    cp /etc/rancher/k3s/k3s.yaml /vagrant_shared
```

[1] https://www.vagrantup.com/
[2] https://k3s.io/
[3] https://www.virtualbox.org/
[4] https://gitlab.com/akosma/k3s-in-vagrant/
[5] https://k9scli.io/

```
    SHELL

agent_script = <<-SHELL
    sudo -i
    apk add curl
    export K3S_TOKEN_FILE=/vagrant_shared/token
    export K3S_URL=https://#{server_ip}:6443
    export INSTALL_K3S_EXEC="--flannel-iface=eth1"
    curl -sfL https://get.k3s.io | sh -
    SHELL

Vagrant.configure("2") do |config|
  config.vm.box = "generic/alpine314"

  config.vm.define "server", primary: true do |server|
    server.vm.network "private_network", ip: server_ip
    server.vm.synced_folder "./shared", "/vagrant_shared"
    server.vm.hostname = "server"
    server.vm.provider "virtualbox" do |vb|
      vb.memory = "2048"
      vb.cpus = "2"
    end
    server.vm.provision "shell", inline: server_script
  end

  agents.each do |agent_name, agent_ip|
    config.vm.define agent_name do |agent|
      agent.vm.network "private_network", ip: agent_ip
      agent.vm.synced_folder "./shared", "/vagrant_shared"
      agent.vm.hostname = agent_name
      agent.vm.provider "virtualbox" do |vb|
        vb.memory = "1024"
        vb.cpus = "1"
      end
      agent.vm.provision "shell", inline: agent_script
    end
  end
end
```

As usual, a `vagrant up` and a `vagrant destroy -f` will simply build and tear down the whole setup, as usual.

In the course of these explorations, I found this bug[6] which basically makes k3s pick the first network interface in the VM, instead of using the one we want; a few command line arguments did the trick.

[6] https://github.com/alexellis/k3sup/issues/306

Kubernetes for Non Technical Readers

Adrian Kosmaczewski

2021-10-01

If you work in the tech field, the word "Kubernetes" is all over the place these days; for those new to the subject, it can be very confusing to understand what it is, what it does, and why it is so important to so many people. In this article I am going to try to go top-down on the whole Kubernetes thing.

Many books and resources about Kubernetes use a bottom-up approach, that is, they start by providing lots of low level definitions, many of which are mostly important for the technically inclined; but here we're going to do things differently.

I'll assume that you have read the word Kubernetes on the press, and you're wondering about what it is, what problems it solves, and how it works. You have some IT background, and maybe you've even worked with backend services and servers in the past. But most importantly, you're curious about this Kubernetes thing, and how it fits in the big picture; hopefully this article will give you some useful pointers. It is, in short, how I would have liked to learn Kubernetes, to have it explained to me.

I've sprinkled the article with plenty of links, so that you can keep exploring after reading it.

TL;DR: sorry, there's none; you have to read the whole article, and it is a long one (~5700 words, ~25 minutes read).

What is Kubernetes?

Simply put, Kubernetes[1] is a platform to run websites and web applications.

And you may ask then, what is a "web application"? It's a program that receives "requests", and returns "responses". You interact with web applications using standard web browsers, such as Firefox[2],

[1] https://kubernetes.io/

[2] https://www.mozilla.org/en-US/firefox/new/

Chrome[3], Safari[4], Opera[5], Brave[6], or Edge[7]. Users can ideally use any browser to use their preferred web apps, and they should (should) get the same experience no matter which one they choose. This is a common experience in 2021.

For example: Jira[8], Confluence[9], WordPress[10], GitLab[11], Basecamp[12], Gmail[13], or even Galaxus[14] are common and popular examples of web applications, that is, they run on a web server, and you use them through a web browser. There is plenty of such applications, but the ones mentioned above are among the most popular of them.

Of course, Kubernetes is not the only way to run web applications, but it has lots of advantages that make it particularly good for some very common scenarios:

- Need for scalability: what if your online shop becomes a hit overnight?
- Cross-platform and portable: what if you want to move your applications from AWS to a cheaper provider, or just run it in your own laptop?
- "Infrastructure as Code" friendly: what if you need to plan for capacity for future expansions, and you need information about your current infrastructure costs? (I'm going to go over the concept of "infrastructure as code" a bit later)
- De facto standard: what if you need new engineers for your team? And can one easily find third-party tools that are compatible with Kubernetes?

For many large applications, Kubernetes provides a very flexible platform for running web applications, and one that has become a standard, for that matter.

But one very important thing to know is that not all web applications need Kubernetes, or even work well with it; for example, some databases systems explicitly aren't Kubernetes-friendly, and engineers must be aware of those limitations.

In those cases, instead of using Kubernetes, you can just run any of these web applications in their own standalone server. By server we mean for example a computer you bought in Digitec[15], one that you

[3] https://www.google.com/chrome/
[4] https://www.apple.com/safari/
[5] https://www.opera.com/
[6] https://brave.com/
[7] https://www.microsoft.com/en-us/edge
[8] https://www.atlassian.com/software/jira
[9] https://www.atlassian.com/software/confluence
[10] https://wordpress.org/
[11] https://about.gitlab.com/
[12] https://basecamp.com/
[13] https://gmail.com/
[14] https://www.galaxus.ch/
[15] https://www.digitec.ch/

have connected yourself to the network in your office, and which you make available in your local network. A web server can also be a virtual machine you run on your own laptop, like when you use Oracle VirtualBox[16], or on AWS EC2[17], for example.

To summarize, Kubernetes is just one way to run web applications, and one that has become really ubiquitous and popular at that.

A Simple Example

Let's imagine you want to watch movies you've (legally) downloaded in the confort of your home.

To do this, you could buy a new small Mac Mini[18], or a Synology NAS[19], and install the Plex media server software[20] in it. Plug a USB disk with your movie collection, and then you can watch those movies from your flatscreen TV in your living room. That's a very simple kind of web application, because your partner can actually access the same Plex application from your laptop in their room, just by using a web browser. The TV is just another "client" of the Plex server; so is the laptop of your partner.

In this simple example, you are running a private service, just for you and your family, in your local network (hopefully you have cabled your home beforehand, or the video quality would suffer a bit). Since there's probably no more than 10 devices (or "clients") in your home accessing the Plex server, it's enough for you to have just one server. Synology NAS units and Mac Minis are quite powerful little machines, as it turns out.

In this case, your TV "requests" a movie, and the Plex server "responds" with the stream of data that contains the audio and video for your enjoyment.

Of course, you're the "sysadmin" of such server; if you need to update your Plex, because there's a new version with new features you'd like to use, you can just update it, and since your data is stored in a separate USB drive, you don't risk losing anything at all. During the time of the update, of course, nobody can watch movies, so you do that on a Saturday morning, or at any other time when nobody is watching movies.

[16] https://www.virtualbox.org/

[17] https://aws.amazon.com/ec2/

[18] https://www.apple.com/mac-mini/

[19] https://www.synology.com/en-global/products?tower=ds_j%2Cds_plus%2Cds_value%2Cds_xs

[20] https://www.plex.tv/media-server-downloads/

History

At the beginning of the World Wide Web[21], most systems ran exactly like the Plex server I've described above. Even Jeff Bezos started Amazon with a single server directly hooked onto an Internet connection. To update software in such a server, developers would just copy the files into the server, verify that everything still worked, and pat themselves on the back when it did.

In my case, when I started working as a software developer in 1997, we had our server (note the singular) in an MCI WorldCom[22] datacenter in upstate New York, south of Albany, in the USA. We did that because Internet connectivity in Buenos Aires was very bad back then, simply because Buenos Aires is far away south, and not many transatlantic optic cables reached the area back then. Bandwidth was narrow, and hosting the server in the USA was a better option.

At the beginning we had just one server there; it ran two important pieces of software:

- A database server, where all the data of our company was stored (if you're curious, it was a SQL Server 6.5[23] database server).
- A web server, which was configured to return pages of information to people connecting to it.

The server was using a Pentium II[24] chip, running a version of Windows called Windows NT[25], with around 256 MB of RAM, which was a crazy expensive amount of memory back then. This simple setup allowed us to serve a whooping 10'000 visitors per day, a record by those standards.

By 2000 we raised some VC money, so we upgraded our installation: we decommissioned the old server, and we bought two new, much more powerful servers; one for the database, and another for the web server, both talking to each other in a simple local network. This allowed us to multiply by 10 the number of visitors we could serve every day, and so we grew.

We uploaded the programs from Buenos Aires for our web application running in the USA using something called FTP (which stands for "File Transfer Protocol"). To manage the system, for example for reboots, backups, and other operations, we used the "Remove Desktop"[26] functionality provided by Microsoft.

We relied on manual operations, and of course, we made mistakes all the time. We had to often roll back operations that went wrong, and

[21] https://www.history.com/news/the-worlds-first-web-site
[22] https://en.wikipedia.org/wiki/MCI_Inc.
[23] https://winworldpc.com/product/sql-server/65
[24] https://en.wikipedia.org/wiki/Pentium_II
[25] https://en.wikipedia.org/wiki/Windows_NT_4.0
[26] https://en.wikipedia.org/wiki/Remote_Desktop_Protocol

restore old backups, all done manually. This was complicated, brittle, and very clumsy.

Problems

At the beginning of the 2000's, the web was becoming more and more popular, and many successful companies had much more traffic than a single server could handle. That is, they could not **scale**.

This is when the concept of a **"load balancer"** appeared; it's a special kind of software that distributes requests among separate, connected, web servers. This meant that you could have lots of web servers, and one load balancer in front of all of them, to distribute requests among all web servers. Above we mentioned nginx[27], which is a common and popular choice among load balancers.

Load balancers use different algorithms to distribute requests among web servers; the simplest one is called the "round robin algorithm", which simply sends one request per server, server after server, in order. Other algorithms take into account the level of load of each server, so that no server is overloaded with heavy-duty requests at any time. There are other more complicated algorithms, and engineers must choose the one best suited for each particular situation.

But, the thing is, when you have lots of web servers running at the same time with the same application behind a load balancer, one specific problem appears: if a web application allows users to log in (a very common requirement, as it turns out), the same user should be logged at once in all web servers behind the load balancer; you can't pretend the user should login in each server every time! That would not be very convenient, and you would lose customers. You must have one login to rule them all.

Whenever a user "logs in" to an application, a **"session"** is created; that's a fancy name for a small piece of data kept in memory, containing the name and other settings belonging to the current user. In such an application, each user sees their own name and preferences in the application, and cannot access other people's settings. But what happens when you have lots of web servers behind a load balancer?

Well, you need another server just to keep session information separate from the web servers. And now, all of a sudden, you have a very complex architecture:

- One or more web servers.
- A load balancer distributing requests among web servers, for example nginx[28], to allow the whole service to scale.
- A security information repository; for example, these days com-

[27] https://nginx.org/en/docs/http/load_balancing.html
[28] https://nginx.org/en/docs/http/load_balancing.html

panies use systems such as OpenLDAP[29] or Keycloak[30] to store that information, to store the usernames and passwords of their users in a secure server.
- A session store; many web applications use a special database called Redis[31] for that.
- And yes, maybe one or more database servers, maybe also behind a special load balancer, just for them!

Here's an important concept: all of these services and servers put together, form what is usually called a **"cluster"**.

And there's plenty more of things that you can have on a cluster: for example, message queues[32] to process purchases; mail servers[33] to send newsletters and forgotten passwords; firewalls to filter requests (usually for security purposes); storage devices connected to the network (like a very expensive Synology NAS) with plenty of disk space available; and much more.

So, remember this: the common name for all of these things is a "cluster". And each of the servers in a cluster is called a **"node"**. Those nodes can be physical or virtual, or a combination thereof, as stated previously.

Complexity

As we just saw, a cluster means having lots of servers to manage, update, backup, and restore. For example, when you have many application servers, and your development team makes an update to your application, you need to roll that new version to all nodes, at once, sharing secrets such as passwords and keys in each node, and without errors!

To make things more difficult, each of the applications we mentioned above is written with different programming languages: for example, GitLab is written in the Ruby programming language[34]; WordPress in PHP[35], while Jira and Confluence are written in Java[36]. Each of these applications required, thus, separate libraries and configurations, that are absolutely and completely incompatible with each other. This means that the upgrade procedures for each language are also different!

An update might also require database migrations, and if anything goes wrong, you need to roll back to the previous version of the appli-

[29] https://en.wikipedia.org/wiki/OpenLDAP
[30] https://www.keycloak.org/
[31] https://redis.io/
[32] https://en.wikipedia.org/wiki/Message_queue
[33] https://whatismyipaddress.com/mail-server
[34] https://www.ruby-lang.org/en/
[35] https://www.php.net/
[36] https://en.wikipedia.org/wiki/Java_(programming_language)

cation as soon as possible, because your customers can jump to your competition in a blink of an eye.

To top it off, you need to keep an eye on all of those servers; are they running properly? Are you sure they aren't running out of memory? How about disk space, is there enough? Can you be notified if any of those servers is down?

And what happens if you need an exact replica of this cluster for your development team? And another for your quality assurance team?

Managing all of this complexity by hand is clearly not a solution for big web applications. We need a better way.

Evolution

In 2005, Amazon pioneered the concept of "Infrastructure as a Service" or "IaaS" when it launched Amazon Web Services (AWS)[37]; instead of having your own hardware servers on a datacenter, just like we had in MCI WorldCom, you can rent a small portion of their infrastructure, available all over the planet in various datacenters with excellent connectivity, and run as many VMs (virtual machines) as required.

The same idea is now available by various providers: Exoscale[38], cloudscale.ch[39], Microsoft Azure[40], Google Cloud[41], IBM Cloud[42], Alibaba Cloud[43], Oracle Cloud[44]; they all provide the same basic building blocks of infrastructure:

- Compute (that is, virtual machines, like Amazon's EC2 service[45])
- Storage (in various forms, but usually compatible with Amazon's S3 service[46])
- Security (users, groups, etc)

Later on, around 2010, there was an explosion of new projects based on the concept of "Infrastructure as Code". In essence, infrastructure as code means writing down a description of your… well, infrastructure, in a machine-readable text file; and then letting a piece of software figure out how to make such infrastructure happen or exist.

You might have heard of Terraform[47], for example; it is used to setup many virtual machines at once, with one command, and to install

[37] https://aws.amazon.com/
[38] https://www.exoscale.com/
[39] https://www.cloudscale.ch/en/
[40] https://azure.microsoft.com/en-us/
[41] https://cloud.google.com/
[42] https://www.ibm.com/cloud
[43] https://eu.alibabacloud.com/en
[44] https://www.oracle.com/cloud/
[45] https://aws.amazon.com/
[46] https://aws.amazon.com/s3/
[47] https://www.terraform.io/

software in them.

This is an example of a Terraform file defining an Exoscale SKS[48] (Scalable Kubernetes Service) cluster hosted in their datacenter located in Geneva, Switzerland:

```
locals {
  zone = "ch-gva-2"
}

resource "exoscale_sks_cluster" "demo" {
  zone          = local.zone
  name          = "demo"
  version       = "1.21.1"
  description   = "Webinar demo cluster"
  service_level = "pro"
  cni           = "calico"
  addons        = ["exoscale-cloud-controller"]
}

resource "exoscale_sks_nodepool" "workers" {
  zone               = local.zone
  cluster_id         = exoscale_sks_cluster.demo.id
  name               = "workers"
  instance_type      = "medium"
  size               = 3
  security_group_ids = [exoscale_security_group.sks_nodes.id]
}
```

A “node pool” as specified above is nothing else than… a group of nodes, or virtual machines, running in the Geneva datacenter.

There’s many other similar systems; suffice to mention Vagrant[49], Puppet[50], Ansible[51], and Chef[52], all of which can be used for similar purposes in different contexts.

Kubernetes provides a form of Infrastructure as Code as well, but specifically not for hardware (physical or virtual, which is something Terraform does very well), but rather for software.

With Kubernetes configuration files you can specify the exact name and version of the applications you want to run, the number of copies running in production, and you can even specify load balancers for it all.

And since everything is written down in a text file, you can create an exact replica of your production environment for development, testing, redundancy, or any other purpose. Just re-run the same software

[48] https://www.exoscale.com/sks/
[49] https://www.vagrantup.com/
[50] https://puppet.com/
[51] https://www.ansible.com/
[52] https://www.chef.io/

on a different environment, and build a copy of your infrastructure somewhere else. Boom, done.

Now we're talking. We can use Terraform to bootstrap a new cluster in Exoscale in less than 5 minutes, commissioning some virtual machines and installing all the required components, and then install and run lots of software on it.

Anatomy of a Kubernetes Cluster

As said previously, Kubernetes is a platform to run web applications. And it does that by managing a cluster, just like the ones we defined above.

A Kubernetes cluster requires nodes, as expected. These nodes are either hardware machines (for example, expensive Dell[53] or Hewlett Packard[54] enterprise servers, or cheaper and smaller Raspberry Pi[55] machines) or virtual machines (like those you can create with VirtualBox on your laptop). Those nodes usually run Linux, but since not so long ago they can also run Windows.

Of all of those nodes you have, Kubernetes takes one (usually the more powerful one) and gives it a special name: the **"Control Plane Node"**. The other nodes, usually cheaper and smaller, are called **"Worker Nodes"**. You usually need at least one control plane node and two worker nodes to have a decent cluster, although just one of each also might be a good choice depending on your needs.

(Turns out you can also load balance the control plane, to create what is usually referred to as an "HA" or "High Availability" cluster, but that's a subject for another time.)

The role of the control plane is to distribute "work" among the worker nodes. The control plane watches the worker nodes continuously, and can decide at any time to "drain" a node and move work to another node.

Hence, the more worker nodes, the better, because each node brings memory and CPU power to the whole cluster. But that also means higher costs, both financial and human, to get everything wired together properly. This is a common trade off.

Developers and system administrators "talk" to the control plane (that is, send requests to it) so that work is distributed, launched, and monitored on all worker nodes. This is done through what is called the Kubernetes API[56] (or Application Programming Interface), that is highly standardized. Developers do not need to know all the details of that API, and just use a tool called `kubectl`[57] to talk to the control

[53] https://www.delltechnologies.com/en-us/servers/poweredge-rack-servers.htm
[54] https://www.hpe.com/us/en/servers.html
[55] https://en.wikipedia.org/wiki/Raspberry_Pi
[56] https://kubernetes.io/docs/concepts/overview/kubernetes-api/
[57] https://kubernetes.io/docs/reference/kubectl/overview/

plane.

There are many other components inside of the control node:

- The API server, mentioned above.
- The scheduler, selecting the nodes in which to run work.
- The controller manager, in charge of managing work and its lifecycle.
- A database called etcd[58] to keep track of the whole inventory of stuff inside of the cluster: work, nodes, configuration, etc, so that if the whole cluster must be rebooted for some reason, everything comes back alive in a similar state.

This means that the control plane is monitoring continuously the worker nodes, so that all apps are running smoothly, and notifies cluster administrators of outstanding events. It can also scale applications, if needed, when the demand grows substantially, or even restart applications if they are not behaving properly. In short, it does a lot of work on behalf of human operators.

Pods, Containers, Deployments, Services, Storage…

We mentioned previously that a control plane distributes "work" among the nodes of your cluster. But what how does that "work" look like?

Simply put, a unit of work in Kubernetes is one or more small containers running your applications inside. Those units of work composed of containers are called **"pods"**.

For example, in the case of GitLab, you can have a single pod running the application, and another pod running a database server. GitLab is written in the Ruby programming language, and has a certain number of "dependencies" it requires to run: configuration files, libraries of code, and more. A pod is a lightweight way to encapsulate the app together with all of its dependencies in a single portable unit. Kubernetes can "schedule" pods to run on a node or on another, depending of many factors, most of which are configurable by the operators of the cluster.

Containers and Image Registries

And what are pods made of? Well, internally pods consist of one or many **"containers"** (well, usually just one, but you could have many if needed). And what are containers? They could be thought of as very lightweight, very fast small virtual machines. And how do you make them? Developers can create and run container images (think "templates") in their machines, without the need for a full Kubernetes cluster, using any of the following applications:

[58] https://etcd.io/

- Docker[59]
- Podman[60]

Once developers create container images, they can share them with other developers, either in the open, or with their colleagues inside an enterprise. For that, developers store containers images inside a **container image repository**, of which there are many examples: Docker Hub[61], Quay[62], AWS ECR[63], GitHub packages[64] (ghcr.io), the Google Container Registry[65], and ttl.sh[66]. Some companies even choose to run their own container image registry internally, using tools like the OpenShift image registry[67], the GitLab Container Registry[68] or Harbor[69].

As I said, developers do not need Kubernetes to run containers; but we do need containers to do useful things with Kubernetes, because pods are made out of containers, and Kubernetes runs pods. This is why Kubernetes is usually referred to as a **"container orchestrator"** system, because it helps running complex apps made with various containers all wrapped into pods, and connected with one another, like the complex system we described earlier in this document.

We are going to talk about containers in another article (and, if you are technically inclined, please forgive my assertion about containers being "small virtual machines" above). For the moment, suffice to say that in the case of Kubernetes, pods "wrap" one or more containers. This allows Kubernetes to move work from node to node more easily. And it is helpful, because some applications require various containers to work together, tightly bound one to the other, so you need to have them always together. A pod provides precisely this advantage.

Deployments

Some applications are more complex and require not just one pod, but many pods at once; for that matter, Kubernetes has the concept of **"deployments"**; a deployment is a set of pods that must always run together.

Kubernetes makes sure that if, by ill luck, one of the pods of a deployment dies (because it crashes or some other problem) another one

[59] https://www.docker.com/
[60] https://podman.io/
[61] https://hub.docker.com/
[62] https://quay.io/
[63] https://aws.amazon.com/ecr/
[64] https://github.com/features/packages
[65] https://cloud.google.com/container-registry/
[66] https://ttl.sh/
[67] https://docs.openshift.com/container-platform/4.5/registry/architecture-component-imageregistry.html
[68] https://docs.gitlab.com/ee/user/packages/container_registry/
[69] https://goharbor.io/

replaces it as soon as possible. This is done automatically by Kubernetes, and operators can use **“liveness probes”** to make sure that Kubernetes checks pods correctly.

There is another kind of probes used by Kubernetes, called **“readiness probes”** which is used to make sure that pods only receive requests when they are ready for them. They are used when the pods start.

We mentioned above that Kubernetes has a form of “infrastructure as code” built in; well, below you can see an example of a Kubernetes deployment defined in a text file using a format called YAML, or “Yet Another Markup Language”, which defines a hierarchy of key and value pairs. Think “ini” files, like those common in old versions of Windows, but with a tree hierarchy.

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: sample-deployment
spec:
  replicas: 3
  template:
    spec:
      containers:
      - image: quay.io/username/container-name:latest
        imagePullPolicy: Always
        ports:
        - containerPort: 9090
          name: sample-port
        resources:
          limits:
            cpu: 150m
          requests:
            cpu: 100m
        livenessProbe:
          httpGet:
            path: /healthz
            port: sample-port
          periodSeconds: 10
          initialDelaySeconds: 10
```

In the example above, the deployment specifies 3 pods, each containing one container inside, based on a container image called `quay.io/username/container-name:latest`, that is, stored in the Quay[70] image registry. The `latest` word indicates the version of the container image; which means that, if needed, we could be using a previous version, in case the newest available has bugs.

The deployment above also shows some interesting limits for the deployment, so that pods don’t consume more CPU time than allocated.

[70] https://quay.io/

This is very important for operators, to be able to control their usage costs, and to plan for future upgrades.

Scaling

Another thing that Kubernetes provides is **automatic vertical scaling**, that is, launching new pods if the demand of a service grows; for example, if your web application store is very popular, and you have lots of new customers online, Kubernetes spins new pods automatically to cope for the demand. Those pods could (should) run in a separate node, if there are any available.

The YAML below defines a simple autoscaling strategy:

```yaml
apiVersion: autoscaling/v1
kind: HorizontalPodAutoscaler
metadata:
  name: sample-autoscaler
spec:
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: sample-deployment
  minReplicas: 1
  maxReplicas: 30
  targetCPUUtilizationPercentage: 20
```

Of course, this might need new nodes to be added, which is something that Kubernetes cannot help you with, because Kubernetes is about software, not hardware; the nodes must be added by an engineer, either manually, or using a tool like Puppet, for example. Kubernetes can, however, once a new node has been added to the cluster, distribute work on it automatically without needing to restart.

Services

Kubernetes does not make applications available to the Internet by default. This might sound counterintuitive, but it's for security reasons; nothing is visible by default. To expose a deployment to the outer world, you must create an object called a **"service"**. There are many kinds of services available in Kubernetes, but the most common one is the "LoadBalancer" kind, which you already now what it is for.

This is how you define a LoadBalancer service in YAML:

```yaml
apiVersion: v1
kind: Service
metadata:
  name: fortune-service
spec:
  type: LoadBalancer
  ports:
```

```
  - port: 3000
    targetPort: fortune-port
```

Storage

Finally, another thing that Kubernetes allows to specify in code is the kind and amount of storage for applications. This allows operation teams to plug and unplug storage providers, and to configure applications to use them accordingly. This way you can allocate 10 GiB of disk space to this application on a fast SSD drive, or 50 GiB of space in a slower but cheaper SATA disk. Some common providers of storage for Kubernetes include Rook[71], Gluster[72], and Longhorn[73].

More

There are many more things that Kubernetes can manage off-the-box: for example, **secrets** (passwords, keys, etc); **configuration values** (which you can change after deployment, if needed); **environment variables** (used to tweak the behaviour of an application in subtle ways); **namespaces** (to separate different objects belonging to different applications running in the same cluster); and much, much, much more.

Installing Applications in Kubernetes

There are many ways to install applications in a Kubernetes cluster:

- The usual way is to write files in YAML format, with all the required options, and then "applying" those files to the cluster using the `kubectl` tool specified above.
- Another way is to combine many YAML files together in a "kustomization file" (yes, with a K), so that they are all applied together in one operation.
- Finally, most apps these days are bundled with what are called "Helm[74] Charts" which are lots of YAML files in a standard structure, specifying all of the pods, deployments, services and storage options required for an application to run.

Lately there is another option to install applications on Kubernetes, using what is called an "Operator[75]", allowing Kubernetes to extend the number of objects available in the cluster. For example, K8up[76] is a backup system bundled as an operator, which allows developers to create YAML files that explicitly mention the "Backup" or "Restore" objects.

[71] https://rook.io/
[72] https://www.gluster.org/
[73] https://rancher.com/products/longhorn/
[74] https://helm.sh/
[75] https://kubernetes.io/docs/concepts/extend-kubernetes/operator/
[76] https://k8up.io/

Flavors of Kubernetes

Kubernetes is an open source project originally created by Google. They based it on a previous system they used internally called "Borg[77]", in production since at least the 2000s. Google donated Kubernetes to the Cloud Native Computing Foundation[78], the CNCF, which is a non-profit that raises money so that developers can keep on working on various projects related to Kubernetes.

Kubernetes is written in the Go programming language[79], which has many characteristics that are specially useful for cloud projects. This is the reason why many of the most popular applications hosted by the CNCF or used around Kubernetes are written in Go, such as Kubernetes, Docker, Prometheus[80], Istio[81], and many, many others.

By the way, the word "Kubernetes" is of Greek origin, and is actually pronounced "Kivernitis"; it means captain or driver of a ship, which is why the logo of the Kubernetes project is the steering wheel of a ship. Kubernetes is the Greek root of the words "cybernetics" and "government", which should sound very appropriate to you by now.

Another way to write "Kubernetes" is to just write "K8s", where the "8" is the number of letters between the "K" and the "s" in "Kubernetes". This is serious, please don't laugh.

Because it's open source, Kubernetes is available to everyone and all, so companies are free to grab the source code, apply their own modifications on top, and sell it if they wish; this is how the following projects came to existence:

- Red Hat OpenShift[82]
- Rancher RKE[83]
- Amazon EKS[84] and EKS Anywhere[85]
- Google GKE[86]
- Microsoft Azure AKS[87]
- Exoscale SKS[88]
- Katacoda[89]
- DigitalOcean Managed Kubernetes[90]

[77] https://static.googleusercontent.com/media/research.google.com/en//pubs/archive/43438.pdf
[78] https://www.cncf.io/
[79] https://golang.org/
[80] https://prometheus.io/
[81] https://istio.io/
[82] https://www.redhat.com/en/technologies/cloud-computing/openshift
[83] https://rancher.com/products/rke/
[84] https://aws.amazon.com/eks/
[85] https://aws.amazon.com/eks/eks-anywhere/
[86] https://cloud.google.com/kubernetes-engine/
[87] https://docs.microsoft.com/en-us/azure/aks/
[88] https://www.exoscale.com/sks/
[89] https://www.katacoda.com/courses/kubernetes/playground
[90] https://www.digitalocean.com/products/kubernetes/

- IBM Cloud Kubernetes Service[91]
- And the smaller members of the family, meant to run in your laptop or on small computers: Minikube[92], MicroK8s[93], Kind[94], and K3s[95]

To avoid confusion, I should add that the "Rancher Enterprise Management System" (or "Rancher" for short) is a different thing[96] than the Rancher RKE project mentioned above; it is a web application that allows you to manage Kubernetes installations, but it is not a flavor of Kubernetes per se. But yes, Rancher can run on a Kubernetes cluster! There's a Helm chart for it, of course (see? All of the concepts are falling in place now). The Rancher Enterprise Management System is extremely popular and convenient, and lots of companies use it to manage their clusters from a central location every day.

The usual analogy to explain the various flavors of Kubernetes is to compare it with the various flavors of Linux distributions. In Linux, there is a project for the Linux Kernel[97] on one side, and then there are "Linux distributions" such as Ubuntu[98], openSUSE[99], Red Hat Enterprise Linux[100] (aka RHEL), each targeting different use cases, on the other. For example, Ubuntu is great for home users, while RHEL is suitable for big enterprises with thousands of employees. Installing the raw Linux kernel can be cumbersome, but Ubuntu and RHEL all come with nice graphical installers that make it easy to install.

Similarly, each Kubernetes flavor has a target audience: K3s is great for hobbyists, while OpenShift is for big enterprises. RKE is great for teams who want to run and manage their own Kubernetes cluster in their own premises, and Minikube is great for developers. They are all much easier to install and launch than the raw Kubernetes project. And so on.

Added Value

Many providers add value to Kubernetes in different ways, usually as follows:

- Pre-configured storage options, ready to use.
- Security: users, groups, roles, etc.
- Built-in CI/CD functionality.
- Simplicity of installation and management.

[91] https://www.ibm.com/cloud/kubernetes-service
[92] https://minikube.sigs.k8s.io/docs/
[93] https://microk8s.io/
[94] https://kind.sigs.k8s.io/
[95] https://k3s.io/
[96] https://rancher.com/products/rancher/
[97] https://www.kernel.org/
[98] https://ubuntu.com/
[99] https://www.opensuse.org/
[100] https://www.redhat.com/en/technologies/linux-platforms/enterprise-linux

- And more…

The reason they do this is because by default, Kubernetes does not provide any of these things; no user interface, no user management, no CI/CD, nothing at all. And to make things worse, it can be complicated[101] to install it from scratch!

The canonical example of such added value is Red Hat OpenShift, which is touted as an enterprise-ready Kubernetes platform. It is 100% compatible with Kubernetes, but provides much more, like integrated user management, a CI/CD engine, "image streams", serverless features, etc.

Both OpenShift and the Rancher Management System provide a visual GUI, which is a full replacement for the YAML files mentioned above; this way, administrators can quickly operate on one or many clusters using a point-and-click user interface. You don't need to write the files; just point and click. Some companies prefer that approach.

The simplest, smallest possible Kubernetes distribution is K3s[102], also created by Rancher, which is targeted for "Internet of Things" and "Edge" deployments, that is, small clusters running in small machines like Raspberry Pi.

There are other Kubernetes distributions that can run on a laptop: Minikube[103], Kind[104], Docker Desktop[105], MicroK8s[106], K3s[107] / K3d[108], and AWS EKS Anywhere[109] are examples. With a single command, a few seconds later you have a small cluster running in your laptop, where the nodes are simulated using virtual machines or containers running locally. This helps developers create applications for Kubernetes, as they can just run them locally without having to pay for an external cluster in a cloud provider.

Conformance

One of the jobs of the CNCF is to verify the conformance[110] of all those Kubernetes flavors, so that they are certified; this is done automatically, through scripts, that verify each and every standard feature of the Kubernetes specification in each flavor of Kubernetes. For example, you can find the conformance results for K3s[111] in GitHub, and be sure that it passes all tests.

[101] https://github.com/kelseyhightower/kubernetes-the-hard-way
[102] https://k3s.io/
[103] https://minikube.sigs.k8s.io/docs/
[104] https://kind.sigs.k8s.io/
[105] https://www.docker.com/products/docker-desktop
[106] https://microk8s.io/
[107] https://k3s.io/
[108] https://k3d.io/
[109] https://aws.amazon.com/eks/eks-anywhere/
[110] https://www.cncf.io/certification/software-conformance/
[111] https://github.com/cncf/k8s-conformance/pulls?q=is%3Apr+k3s

This ensures that an application that could run on Minikube can also run on EKS; of course, this is not true the other way around. First, because your laptop running Minikube is much less powerful than the servers at AWS. And second, because many distributions extend Kubernetes with proprietary extensions. This means that an application specifically built with OpenShift in mind might not (and usually does not) work properly in K3s. Of course, doing so might provide economic benefits, but it also reduces portability; but this is another common tradeoff in our industry.

By using a certified Kubernetes distribution, organizations can be sure to find certified engineers, compatible tooling, and thus reduce the risk of lock-in. Theoretically, you are able to move your application from one provider to another with minimum effort, and that in itself is a strong incentive for businesses to use Kubernetes.

What's Next?

If you read until here, congratulations and thanks! There is so much material about Kubernetes that it is hard to decide where to go.

The links in the article might serve as guides for you to learn more about it, but if what you want is to get your hands dirty and "make your own cluster", you should watch this video: "i built a Raspberry Pi SUPER COMPUTER!! // ft. Kubernetes (k3s cluster w/ Rancher)"[112] by NetworkChuck (published July 2021). It is very funny, very well explained, and it actually teaches you how to create a cluster using the K3s[113] Kubernetes distribution and one or many Raspberry Pi[114] computers.

[112] https://www.youtube.com/watch?v=X9fSMGkjtug
[113] https://k3s.io/
[114] https://en.wikipedia.org/wiki/Raspberry_Pi

Git for Non Technical Readers

Adrian Kosmaczewski

2021-10-08

If you are in the business of software, sooner or later you will hear people talking about Git, GitHub, or GitLab. What are they? To explain that, we must learn what Git[1] is first.

What is Git?

> Git is a free and open source distributed version control system designed to handle everything from small to very large projects with speed and efficiency.

There are several things in the definition above that make absolutely no sense to people that aren't that much into computers.

- "Free and open source": that's probably the only part that is actually easy to understand; it means that the tools are freely available to all, without cost or obligations.
- "Distributed": there's an idea of locality, of space here.
- "Version control": clearly something to do with those "versions" that all software packages mention in their websites.

So the complicated parts here are "distributed" and "version control". Let's start with the last one first.

Managing File Versions

All of us have experienced the typical collaboration project, where many people work together on the same Word or Photoshop file. First the file is called `draft-1.psd`, then comes `final-edited-by-john.docx`, then it becomes `draft-for-comments-14.xlsx`, and next thing you know, contributions are scattered among tens of files and nobody actually knows what the "actual" final document is.

Of course these days there are online collaboration tools such as Google Docs, Confluence, or Office 365 that allow several people to edit the same file at once, and you can see their cursors on the screen as they write. Still, even with this capability, we might end up with several versions of files laying around.

[1] https://git-scm.com/

Thankfully many of these tools offer a “history” function showing a list of versions, and their respective author and creation date. Well, Git does precisely something like this.

Software

In the world of software, source code is the basic unit of collaboration. Source code is just text written in some language, which is then used to “build” the final “thing” that is installed in a PC or in a server, like Word, or this very website. For a single software project, you can have hundreds of different files, sometimes even hundreds of thousands of them.

Needless to say, keeping track of it all is very, very complicated without help.

But early on, developers realized that, when working in a software package consisting of many files, it is very rare that two developers are editing the same line of the same file at once.

Hence, it makes sense to allow people to work together, since the actual chance of collision is really, really low.

And if developers need to work together in the same problem, they might want to do it on the computer of one of them, sharing the keyboard and exchanging ideas (or, these days, over Zoom) which removes any chance of conflict. This technique, by the way, is usually referred to as “Extreme Programming”[2] or “XP”.

Version Control

A “Version Control System” provides teams with the ability to keep every change made, to every file, in every folder, of every project, forever. Every change in the project is stored along with the following information:

- **Who**: the name and e-mail of the person that made the commit
- **When**: the date and time of the change
- **What**: the changes themselves

About the “what”, here’s an important point: to be efficient, version control systems do not store a complete file every time that it changes; but only the lines that changed. It seems complicated (it is, to a certain degree) but the core concept is, precisely, this.

And Git is, essentially, just another Version Control System.

Terminology

Every project in a version control system is called a “repository”. A repository contains all the files in a project, stored within folders. The

[2] https://en.wikipedia.org/wiki/Extreme_programming

version control system allows users to see more than just the files; it allows to see the “history” of each files, and also to “blame” (ugly word, indeed) and see who wrote any line of code in any file of the repository.

Remember the “history” feature I mentioned before? Well, here it is.

This tracking allows developers to understand the evolution of the project with the finest possible grain.

When developers start working in a project, they “check out” a copy of the project on their own computer. Through the network, they receive the files, and can work on them: add features, solve bugs, add documentation, even execute tests, or performing some management task like counting the lines of code in the project.

Whenever developers make a modification that works (that is, a new feature or a fixed bug) they “commit” the change. This adds a new entry in the history of the project.

Every so often, developers can “tag” a particular commit, so that they can find faster a previous version. For example “v1.0” would be a very useful and valid tag. You can tag projects any way you want, with words or numbers. Anything goes, really.

History

Others have written about the history of version control systems, so I’m just going to link to them here:

- Eric Sink[3].
- Lynda[4]
- Redgate[5]

For simplicity, I like to group the history of version control systems in three generations.

First Generation: No Networking Support, and Single-user

The first version control systems were designed during the 1970s; the need to collaborate on source code clearly dates from the early times of computers.

Two important early systems are:

- SCSS[6] released in 1973, written in SNOBOL for the IBM System/370 running OS/360 (closed source until 2005)
- GNU RCS[7] started in 1982, still maintained! (free, open source)

[3] https://ericsink.com/vcbe/html/history_of_version_control.html

[4] https://www.lynda.com/ALMTFS-tutorials/history-version-control/106788/115979-4.html

[5] https://www.red-gate.com/blog/database-devops/history-of-version-control

[6] https://en.wikipedia.org/wiki/Source_Code_Control_System

[7] https://www.gnu.org/software/rcs/

These systems did not cope well with many users editing the same file, which was a major drawback.

Second Generation: Centralized, and Multi-user or "Concurrent"

During the 1980s, computers starting getting connected in networks, which meant that teams could keep a centralized repository, and would check out and commit changes from their own computers.

Many very influential systems were built around this model.

- CVS[8] started in 1986 (free, open source) => the "C" stands for "Concurrent"
- Subversion[9] started in 2000 (free, open source) as a "better CVS" with transactional commits.
- Microsoft SourceSafe[10] (commercial, closed source, early 90s)
- Microsoft Team Foundation Server[11] (Microsoft, today "Azure DevOps Server") (commercial, closed source, 2010)
- IBM Rational ClearCase[12] (commercial, closed source, mid-90s) => great reputation, very high cost
- SourceGear Vault[13] (commercial, closed source, late 90s) marketed as a safer alternative to SourceSafe, targeting the same demographic.
- Perforce[14] (commercial, closed source, 1995) still very much used inside Google.

Centralized systems use a simple architecture: a server stores a database with all the source code, while clients connect and get the "current copy", usually the latest.

These systems had a terrible Achiles' Heel: if the server was corrupted, all the project was lost. This was unfortunately very common with Microsoft SourceSafe, who earned a terrible reputation back in the 1990s. Of course this requires a backup strategy.

Third Generation: Distributed

- BitKeeper[15] (free, open source, 2000) open source since 2016, commercial before.
- GNU arch[16] (free, open source, 2001) now deprecated.
- GNU Bazaar[17] (free, open source, 2005)

[8] http://savannah.nongnu.org/projects/cvs
[9] http://subversion.apache.org/
[10] https://en.wikipedia.org/wiki/Microsoft_Visual_SourceSafe
[11] https://azure.microsoft.com/en-us/services/devops/server/
[12] https://www.ibm.com/products/rational-clearcase
[13] https://www.sourcegear.com/vault/
[14] https://www.perforce.com/products/helix-core
[15] https://www.bitkeeper.org/
[16] https://en.wikipedia.org/wiki/GNU_arch
[17] https://bazaar.canonical.com/en/

- Git[18] (free, open source, 2005) created by the Linux team
- Mercurial[19] (free, open source, 2005) which is losing popularity lately while Git grows
- Fossil[20] (free, open source)

Git is the de facto standard today. All the others are niche, used in very small segments of the market. From that perspective, they are almost nonexistent.

With Git, there is no server anymore; every member of a software development team has a full copy (a "clone") of all the history of all the project. Clones can share changes with one another without distinction. Of course this makes it very, very hard for a malfunction to bring down the whole project; there's plenty of backups in every computer that cloned the repository.

Speaking about terminology, when joining a new project, developers must first "clone" its repository. Then they "commit" changes locally, and after that, they "push" those changes to the source. Every so often they "pull" changes, to get all the changes done by their colleagues.

This is exactly the model used at many companies today.

As a side note, it is important to point out that CVS allowed users to import RCS projects; Subversion could import CVS repositories. And finally, Git allows to import Subversion repositories. This is exactly the path that many projects followed since the 1980s.

GitLab, GitHub, BitBucket…

Even if Git is distributed, developers usually keep a "designated clone" at the center of their collaboration. This requires a centralized server where the "master clone" resides.

GitHub was among the first services to provide such a functionality, and one of the first to do it based on Git. Just like with any other SaaS, users can sign up, create a repository, and then clone it and push changes to it.

A GitHub project is nothing else than a Git project; this means that users can "clone" the project away from GitHub and host it into their own comptuters, or even into a competitor of GitHub, and life goes on.

Given the success of GitHub (eventually bought by Microsoft[21]) came BitBucket[22] (from Atlassian) and finally GitLab[23], which can be self-

[18] https://git-scm.com/
[19] https://www.mercurial-scm.org/
[20] https://www.fossil-scm.org/
[21] https://news.microsoft.com/announcement/microsoft-acquires-github/
[22] https://bitbucket.org/
[23] https://gitlab.com/

hosted. This means that it can be installed “on premises”, which makes it very popular with enterprises. Another common self-hosted system of the same kind is Gitea[24].

All of these options provide many features, complementing the mere fact of just storing repositories:

- Web-based interface accessible from any operating system and web browser.
- A terminal (CLI) or mobile application.
- Repository creation, grouping, management.
- History review and exploration for repositories.
- Continuous integration and deployment (CI/CD).
- Integrated wiki (like Confluence[25]).
- Ticket management (like Jira[26]).
- Project management (milestones, etc).
- Integration with other tools, such as Kubernetes[27] or Terraform[28].
- User rights management.
- Analytics & statistics.

Other Tools for Git

Git being an open source tool, there are lots and lots of tools that you can use to browse, review, manage, and interact with Git repositories. Among the most popular we can find the following:

- Atlassian SourceTree[29] for Windows and macOS.
- Tower[30] for Windows and macOS.
- TortoiseGit[31] for Windows
- Visual Studio Code[32] for Windows, Mac, & Linux, is one of a myriad of text editors with built-in Git support. It has plenty of extensions, among which I can recommend the following two related to Git:
  - Git Graph[33].
  - Annotator[34]

[24] https://gitea.com/
[25] https://www.atlassian.com/software/confluence
[26] https://www.atlassian.com/software/jira
[27] https://kubernetes.io/
[28] https://www.terraform.io/
[29] https://www.sourcetreeapp.com/
[30] https://www.git-tower.com/mac
[31] https://tortoisegit.org/
[32] https://code.visualstudio.com/
[33] https://marketplace.visualstudio.com/items?itemName=mhutchie.git-graph
[34] https://marketplace.visualstudio.com/items?itemName=ryu1kn.annotator

FFmpeg Tips and Tricks

Adrian Kosmaczewski

2021-10-15

Here's a small selection of cool things you can do on Linux with FFmpeg[1].

Convert Between Video Formats

The most basic, simple, and useful thing to do with FFmpeg:

```
$ ffmpeg -i input.m4v output.mp4
```

For example, to convert MOV to MP4

```
$ ffmpeg -i movie.mov -vcodec copy -acodec copy out.mp4
```

You can also use Handbrake[2] to drive these conversions through a GUI.

Create Animated GIFs

There are two ways to do this: first by generating the frames in a subfolder, and then concatenating everything with `convert`:

```
$ ffmpeg -i video.mp4 -r 5 'frames/frame-%03d.jpg'
$ convert -delay 20 -loop 0 frames/*.jpg myimage.gif
```

Where `-r` in the first command stands for the frames per second caught in the GIF.

But can do this without `convert` in one shot:

```
$ ffmpeg -i portal.mp4 -r 5 portal.gif
```

Trim Video

Generate a subclip of a video, removing frames before and after some reference points:

```
$ ffmpeg -i input.mp4 -ss 00:01:15 -t 00:00:30 -async 1 output.mp4
```

- The `-ss` parameter is the starting point.
- The `-t` provides the length of the clip.

[1] https://ffmpeg.org/
[2] https://handbrake.fr/

Make Video Smaller

Smaller in bytes:

```
$ ffmpeg -i input.mp4 -vcodec libx265 -crf 20 output.mp4
```

Smaller in width and height:

```
$ ffmpeg -i input.avi -s 720x480 -c:a copy output.mkv
```

Make a Video compatible with QuickTime

This should make a video that is Mac-friendly:

```
$ ffmpeg -i input.webm -f mp4 -vcodec libx264 \
         -pix_fmt yuv420p output.mp4
```

Make Videos Compatible with Windows Media

This creates an AVI file that is compabible... with Windows 95!

```
$ ffmpeg -i input.mp4 -vcodec msvideo1 -acodec adpcm_ms \
         -vf scale=320:240 -f avi output.avi
```

- The `scale` information is required, and width and height must be multiples of 4 for the `msvideo1` encoder.
- The `avi` parameter selects the container format.

Split Video

As the title implies:

```
$ ffmpeg -i largefile.mp4 -t 00:50:00 -c copy smallfile1.mp4 \
       -ss 00:50:00 -c copy smallfile2.mp4
```

Concatenate Video Files

The inverse operation as above; create a `playlist.txt` file with this structure:

```
# Playlist
file 'part-1.mkv'
file 'part-2.mkv'
file 'part-3.mkv'
```

And then concatenate all movies together:

```
$ ffmpeg -f concat -safe 0 -i playlist.txt -c copy output.mkv
```

To make sure the audio works in QuickTime / iOS when using a playlist:

```
$ ffmpeg -f concat -safe 0 -i playlist.txt -c:v copy -c:a aac output.mp4
```

Better Concatenation

The script below performs a better concatenation using intermediate files.

```
#!/usr/bin/env bash

# This script merges all video files into one.
# It requires ffmpeg:
# Linux: sudo apt install ffmpeg
# Mac: brew install ffmpeg

ffmpeg -i part0.mp4 -c copy -bsf:v h264_mp4toannexb -f mpegts int0.ts
ffmpeg -i part1.mp4 -c copy -bsf:v h264_mp4toannexb -f mpegts int1.ts
ffmpeg -i part2.mp4 -c copy -bsf:v h264_mp4toannexb -f mpegts int2.ts
ffmpeg -i part3.mp4 -c copy -bsf:v h264_mp4toannexb -f mpegts int3.ts
ffmpeg -i part5.mp4 -c copy -bsf:v h264_mp4toannexb -f mpegts int5.ts
ffmpeg -i part6.mp4 -c copy -bsf:v h264_mp4toannexb -f mpegts int6.ts
ffmpeg -i "concat:int0.ts|int1.ts|int2.ts|int3.ts|int5.ts|int6.ts" \
       -c copy -bsf:a aac_adtstoasc output.mp4
rm *.ts
```

Concatenating video files requires the audio to be in the same format, for example in AAC:

```
$ ffmpeg -i input.mp4 -c:v copy -c:a aac output.mp4
```

Record your Webcam

This command records your webcam and generates a video file:

```
$ ffmpeg -f oss -i /dev/dsp -f video4linux2 -s 320x240 \
         -i /dev/video0 out.mpg
```

The `/dev/video0` part should match your own webcam; to find it, for example if your laptop has an integrated webcam but you also have a USB Logitech camera connected:

```
$ sudo apt install v4l-utils
$ v4l2-ctl --list-devices

    Logitech Webcam C930e (usb-0000:00:14.0-3.1):
        /dev/video2
        /dev/video3

    Integrated Camera: Integrated C (usb-0000:00:14.0-8):
        /dev/video0
        /dev/video1
```

Update, 2022-09-30: You can download and merge all the files referenced in a M3U8 video stream into a single file with this command: `ffmpeg -i "http://example.com/stream.m3u8" -c copy`

`-bsf:a aac_adtstoasc "output.mp4"`(source[3]), or you can just use `youtube-dl` for that.

Update, 2023-02-24: Extract the audio track from a video using `ffmpeg -i input.mp4 -vn -acodec copy output.aac`.

Update, 2023-05-19: For lossless and super fast conversion between AVI and MP4, just use `ffmpeg -i input.avi -c:v copy -c:a copy -y output.mp4`

[3] https://windowsloop.com/download-m3u8-video-with-ffmpeg/

Steve Jobs vs Bill Gates on Stage

Adrian Kosmaczewski

2021-10-22

I have had the chance to attend keynotes by Bill Gates and Steve Jobs in person; their styles couldn't have been more different. Here's some memories from both. Of course I did not meet or talk to them; this is just my experience as another attendee in the room.

Bill Gates

I saw Bill Gates in person at the opening keynote of the Office System Developers Conference in 2006, in the Microsoft campus in Redmond, not far from Seattle. You can read the whole speech online[1].

His keynote was, by the way, presented by Steven Sinofsky[2] himself. If you do not know who he is, you should read his excellent memories of his time at Microsoft[3]; it's an outstanding series with lots of anecdotes about the "golden age" of Microsoft at the beginning of the 1990s.

This was shortfly before Bill Gates' last day at Microsoft[4] in January 2008, when he left the company to take care of this foundation[5].

The memory I have of that speech was that of a person deeply uncomfortable on stage. It really looked like Bill didn't want to be there, talking, at that moment. His mind was definitely somewhere else. His words sounded quite fake, dry, even dull. I remember I was at the second or third row from the front, and of course it was quite a moment to be there. It's not like one watches a historical figure like Bill Gates in person every day.

There's one quote that remember in particular, that made the whole room cough and shrug in discomfort:

> We've got a new organization we're announcing called openXMLDeveloper.org. No organization is good unless you put the word "open" right at the front, so we've got it

[1] https://news.microsoft.com/speeches/bill-gates-microsoft-office-system-developers-conference-2006/

[2] https://twitter.com/stevesi/

[3] https://hardcoresoftware.learningbyshipping.com/archive?sort=new

[4] https://www.youtube.com/watch?v=i1M-IafCor4

[5] https://www.gatesfoundation.org/

> right out there. In fact, you know, these are three of my favorite words, “open,” “XML” and “developer,” and that’s all in one organization.

This refers to the time when they announced the opening of the DOCX and XLSX formats. All in all, that conference in itself was quite forgettable. I remember I was expecting to see .NET languages like C# to be finally available in Excel and Word, but nothing like that ever happened.

At that time I was already planning my migration out of the Microsoft galaxy[6]. I was tired of the ecosystem, tired of good old Visual Basic, and of everything related to Microsoft.

Steve Jobs

I saw Steve Jobs on stage twice, during WWDC 2008[7], when he introduced the iPhone 3G (Engadget’s live blog[8]), and WWDC 2010[9], when it was the turn of the iPhone 4 (Engadget again[10]). Both keynotes took place in the big room upstairs in the Moscone Center in San Francisco.

I don’t think it is needed to say how different his style was from Bill Gates’. This was a completely different galaxy. Steve was able to actually make you feel like you had to have those things, that Apple was five years ahead from everything else in the whole world.

As you can see in the videos linked above, Steve was thin but not that much in 2008, but in 2010 his figure was shocking. He looked really thin and frail.

What I remember most from his 2010 keynote (aside from the whole Gizmodo affair[11]) was when he wanted to demo something on the iPhone 4, but the wifi was jammed due to the interference generated by literally thousands of people blogging and tweeting with Mifi devices[12] in the room.

After asking for a little help, he really got quite upset.

> Now before I begin number 6, our guys were running like crazy backstage as you might imagine (laughter), and we figured out why my demo crashed. Because there are 570 wifi base stations operating in this room. OK? We can’t deal with that. So, we have two choices. Either… I got some more demos that are really great that I’d like to show you, so we either turn off all the stuff, and we see the demos or

[6] /blog/the-developer-guide-to-migrate-across-galaxies/

[7] https://www.youtube.com/watch?v=4yVQJ6jJPak

[8] https://www.engadget.com/2008-06-09-steve-jobs-keynote-live-from-wwdc-2008.html

[9] https://www.youtube.com/watch?v=vis7RBCq9ow

[10] https://www.engadget.com/2010-06-07-steve-jobs-live-from-wwdc-2010.html

[11] https://gizmodo.com/this-is-apples-next-iphone-5520164

[12] https://en.wikipedia.org/wiki/MiFi#Radio_interference_at_trade_shows

> we give up and I don't show you the demos. Would you like to see the demos or not? (Cheers) OK, so here's the deal; let's turn off the lights in the hall–several hundreds of these are Mifi devices by the way, so all you bloggers need to turn off your base stations, turn off your wifi, every notebook, I'd like you to put them down on the floor, and all of you, I'd like you to look around and police each other (laughter).

I wanted to see the demos, so I actually asked people around me to comply to his request, and of course I was met with angry looks, some expletive, and then nothing. Some did close their laptops, indeed, but not the majority.

Steve was a great speaker, he was able to drive around such a mess of a situation and deliver a great event. Not all speakers are able to do that.

You can watch the whole sequence in a video online[13], or read about it at a GigaOM article[14].

[13] https://www.youtube.com/watch?v=zduCdHklkAI

[14] https://gigaom.com/2010/06/07/steve-jobs-survives-gizmodo-but-not-mifi/

Lots of VSCode Extensions

Adrian Kosmaczewski

2021-10-29

The recent release[1] by Microsoft of vscode.dev[2], the online version of Visual Studio Code[3], made me think of all the different things I do with VSCode, including this blog. And of course, being productive in VSCode means, to a large extent, finding gems across a seemingly infinite number of extensions.

I love editors, and I love VSCode as well. I have written a whole book[4] about my obsession with Vim and Emacs. Consider this article as an extension of that book.

I started using VSCode relatively late, though. I didn't use it at all, I think, until late 2017. Then I got a job at a company where I got to use it a lot, for various reasons, and I liked many things about it: it's cross-platform, very fast, and of course, it has an amazing number of available extensions. A lot.

I have used each one of the extensions I mention in this article at some point in the past four years, for an extended period of time. Of course I have tried many more, but these are the ones that stuck in my workflow. They are classified in three simple groups, representing each of the things I do with VSCode every day.

Writing Prose

First and foremost, what I do with VSCode is… writing prose. I have found this editor to be a fantastic tool to get words on the screen.

Most of what I write is in AsciiDoc, particularly at my job, and of course I use for that the excellent official extension[5] by Asciidoctor[6].

But of course there is a lot of Markdown in my way; after all, this very blog is generated with Hugo[7] from Markdown sources. And the

[1] https://code.visualstudio.com/blogs/2021/10/20/vscode-dev

[2] https://vscode.dev/

[3] https://code.visualstudio.com/

[4] /books/Tales_Of_Editors_And_Keyboards/Tales_Of_Editors_And_Keyboards.pdf

[5] https://marketplace.visualstudio.com/items?itemName=asciidoctor.asciidoctor-vscode

[6] https://asciidoctor.org/

[7] https://gohugo.io/

Markdown All in One[8] extension is simply perfect for that. I use it to generate (and keep updated) the table of contents, like the one at the top of this article.

Besides AsciiDoc and Markdown, I also enjoy writing screenplays in the Fountain[9] markup language as a hobby, and the Better Fountain[10] extension allows me to write, preview, and export to PDF those exercises in futility. I mean, I'm not going to Hollywood anytime soon. Moving on.

I don't write that much LaTeX anymore, but for the few times I needed a good editor, LaTeX Workshop[11] was a fantastic choice.

For all of these formats, however, it makes sense sometimes to keep an eye on the length; for that, the Read Time[12] extension, which can be easily customized, and the Word Count[13] extension, both provide live feedback about the length of my writing.

I have also installed the Insert Date String[14], the Lorem Ipsum[15], the Trailing Spaces[16], and the Paste URL[17] extensions. This last one is awesome, and I contributed[18] AsciiDoc support to it last year.

For a full-screen, distraction-free editing experience, no need for an extension: just CTRL+K Z and write your heart out.

Programming

I still do quite a bit of programming, even though it is not anymore my primary activity. And VSCode is, as expected, a great platform for this. Whether I write code in Go[19], Rust[20] (with the rust analyzer[21] language server), Kotlin[22], COBOL, REXX, JCL, PL/I[23], F#[24], C, C++[25],

[8] https://marketplace.visualstudio.com/items?itemName=yzhang.markdown-all-in-one
[9] https://fountain.io/
[10] https://marketplace.visualstudio.com/items?itemName=piersdeseilligny.betterfountain
[11] https://marketplace.visualstudio.com/items?itemName=James-Yu.latex-workshop
[12] https://marketplace.visualstudio.com/items?itemName=johnpapa.read-time
[13] https://marketplace.visualstudio.com/items?itemName=ms-vscode.wordcount
[14] https://marketplace.visualstudio.com/items?itemName=jsynowiec.vscode-insertdatestring
[15] https://marketplace.visualstudio.com/items?itemName=Tyriar.lorem-ipsum
[16] https://marketplace.visualstudio.com/items?itemName=shardulm94.trailing-spaces
[17] https://marketplace.visualstudio.com/items?itemName=kukushi.pasteurl
[18] https://github.com/kukushi/PasteURL/pull/11
[19] https://marketplace.visualstudio.com/items?itemName=golang.Go
[20] https://marketplace.visualstudio.com/items?itemName=rust-lang.rust
[21] https://marketplace.visualstudio.com/items?itemName=matklad.rust-analyzer
[22] https://marketplace.visualstudio.com/items?itemName=mathiasfrohlich.Kotlin
[23] https://marketplace.visualstudio.com/items?itemName=IBM.zopeneditor
[24] https://marketplace.visualstudio.com/items?itemName=Ionide.Ionide-fsharp
[25] https://marketplace.visualstudio.com/items?itemName=ms-vscode.cpptools

Smalltalk[26], C#[27], Makefile[28], Python[29] (with the Pylance[30] language server), PHP[31], Java[32] (with Red Hat's Quarkus[33] framework and Microsoft's Debugger for Java[34]), YAML[35], or Shell scripts[36], there's always an extension to help.

Let's not forget the wonderful built-in support for TypeScript[37], including debugging and everything else one might expect. But one thing that is missing is a proper way to run Mocha tests directly from VSCode; the Mocha Test Explorer[38] and the Test Explorer UI[39] provide exactly that.

And speaking about built-in tools, even though VSCode's Git integration is brilliant, I often use Git Graph[40] to visualize the branches of a repository, and Annotator[41] to find out who wrote what.[42].

If you are like me and you try to use Firefox as much as possible instead of other browsers, you're going to love the Debugger for Firefox[43] extension.

It goes without saying that the Terraform[44], Docker[45], Kubernetes[46], and OpenShift[47] extensions are fundamental for any Cloud Native developer. But even better, with the Remote - Containers[48] extension,

[26] https://marketplace.visualstudio.com/items?itemName=leocamello.vscode-smalltalk

[27] https://marketplace.visualstudio.com/items?itemName=ms-dotnettools.csharp

[28] https://marketplace.visualstudio.com/items?itemName=ms-vscode.makefile-tools

[29] https://marketplace.visualstudio.com/items?itemName=ms-python.python

[30] https://marketplace.visualstudio.com/items?itemName=ms-python.vscode-pylance

[31] https://marketplace.visualstudio.com/items?itemName=felixfbecker.php-debug

[32] https://marketplace.visualstudio.com/items?itemName=redhat.java

[33] https://marketplace.visualstudio.com/items?itemName=redhat.vscode-quarkus

[34] https://marketplace.visualstudio.com/items?itemName=vscjava.vscode-java-debug

[35] https://marketplace.visualstudio.com/items?itemName=redhat.vscode-yaml

[36] https://marketplace.visualstudio.com/items?itemName=timonwong.shellcheck

[37] https://www.typescriptlang.org/

[38] https://marketplace.visualstudio.com/items?itemName=hbenl.vscode-mocha-test-adapter

[39] https://marketplace.visualstudio.com/items?itemName=hbenl.vscode-test-explorer

[40] https://marketplace.visualstudio.com/items?itemName=mhutchie.git-graph

[41] https://marketplace.visualstudio.com/items?itemName=ryu1kn.annotator

[42] Git Lens is a very popular extension for Git, and I tried it at some point early on, but I found it so annoying and heavy that I ended up never installing again.

[43] https://marketplace.visualstudio.com/items?itemName=firefox-devtools.vscode-firefox-debug

[44] https://marketplace.visualstudio.com/items?itemName=HashiCorp.terraform

[45] https://marketplace.visualstudio.com/items?itemName=ms-azuretools.vscode-docker

[46] https://marketplace.visualstudio.com/items?itemName=ms-kubernetes-tools.vscode-kubernetes-tools

[47] https://marketplace.visualstudio.com/items?itemName=redhat.vscode-openshift-connector

[48] https://marketplace.visualstudio.com/items?itemName=ms-vscode-remote.remot

one can debug applications running inside a container directly from VSCode. And if you're into K3s[49], get the Kubernetes k3d[50] extension right away.

For those REST APIs that come out of those experiments, the Thunder Client[51] and the Swagger Viewer[52] extensions are always handy. And if a database is around, this Database Client[53] is perfect.

Attentive readers will have noticed that GitHub Copilot does not appear in the list. I do not use it, nor I recommend it.

And More

Coding is the most obvious thing to do with VSCode, but there's so much more you can do with it. Like drawing, for example, either vector images in SVG or PNG formats with the Draw.io Integration[54], or other raster formats with Luna Paint[55]. Think of it as having GIMP and Inkscape integrated into VSCode (or Illustrator and Photoshop, if that's your thing). If you don't need a fully fledged SVG editor, maybe Svg Preview[56] will be just enough.

Another thing you can do is editing files in hexadecimal with the Hex Editor[57]. Or viewing Excel and CSV files with the Excel Viewer[58]. Or creating and running Jupyter[59] notebooks.

Whatever you do, sprinkle your projects with `TODO`, `FIXME` and other tags, and keep track of them all in one place with the Todo Tree[60] extension.

You can customize the colors of your editor with Peacock[61], providing a separate color style for each project. This is a great feature, particularly if you have several VSCode windows open side-by-side. And add some nice icons to the sidebar with the vscode-icons[62] bundle.

---

e-containers

[49] https://k3s.io/

[50] https://marketplace.visualstudio.com/items?itemName=inercia.vscode-k3d

[51] https://marketplace.visualstudio.com/items?itemName=rangav.vscode-thunder-client

[52] https://marketplace.visualstudio.com/items?itemName=Arjun.swagger-viewer

[53] https://marketplace.visualstudio.com/items?itemName=cweijan.vscode-database-client2

[54] https://marketplace.visualstudio.com/items?itemName=hediet.vscode-drawio

[55] https://marketplace.visualstudio.com/items?itemName=Tyriar.luna-paint

[56] https://marketplace.visualstudio.com/items?itemName=SimonSiefke.svg-preview

[57] https://marketplace.visualstudio.com/items?itemName=ms-vscode.hexeditor

[58] https://marketplace.visualstudio.com/items?itemName=GrapeCity.gc-excelviewer

[59] https://marketplace.visualstudio.com/items?itemName=ms-toolsai.jupyter

[60] https://marketplace.visualstudio.com/items?itemName=Gruntfuggly.todo-tree

[61] https://marketplace.visualstudio.com/items?itemName=johnpapa.vscode-peacock

[62] https://marketplace.visualstudio.com/items?itemName=vscode-icons-team.vscode-icons

And, whatever you do, the Live Share[63] extension makes it extremely easy to do some remote Extreme Programming in these times of pandemic, confinement, and work from home.

Update, 2021-12-17: two more I've been using recently; Better TOML[64] and EditorConfig for VS Code[65].

Update, 2022-02-25: add Error Lens[66] to the list. Awesome.

Update, 2022-06-03: Grammarly[67] has become a must have.

Update, 2022-07-22: Diff[68] has saved the day.

Update, 2022-11-08: Just discovered the vscode-pets[69] extension and there goes my productivity. And I also discovered the VS Code Can Do That?[70] website, and it's worth a visit.

Update, 2023-02-24: Add Deno[71] and the Microsoft Edge Tools for VS Code[72] to the mix.

Update, 2023-09-01: Now that Visual Studio for Mac has been retired[73], we have the C# Dev Kit[74] as a replacement.

[63] https://marketplace.visualstudio.com/items?itemName=MS-vsliveshare.vsliveshare

[64] https://marketplace.visualstudio.com/items?itemName=bungcip.better-toml

[65] https://marketplace.visualstudio.com/items?itemName=EditorConfig.EditorConfig

[66] https://marketplace.visualstudio.com/items?itemName=usernamehw.errorlens

[67] https://marketplace.visualstudio.com/items?itemName=znck.grammarly

[68] https://marketplace.visualstudio.com/items?itemName=fabiospampinato.vscode-diff

[69] https://marketplace.visualstudio.com/items?itemName=tonybaloney.vscode-pets

[70] https://vscodecandothat.com/

[71] https://marketplace.visualstudio.com/items?itemName=denoland.vscode-deno

[72] https://marketplace.visualstudio.com/items?itemName=ms-edgedevtools.vscode-edge-devtools

[73] https://devblogs.microsoft.com/visualstudio/visual-studio-for-mac-retirement-announcement/

[74] https://devblogs.microsoft.com/visualstudio/announcing-csharp-dev-kit-for-visual-studio-code/

Polyglot Conway

Adrian Kosmaczewski

2021-11-05

My personal project during the pandemic was Conway[1], a project providing implementations of Conway's Game of Life[2] in as many programming languages as possible.

I like to see how the same idea looks in a different programming language; I like to see the ecosystem around those languages as well: libraries, unit testing, documentation, code formatting and linting, etc.

It started as a teaching tool. I used the first version of it, initially in just TypeScript[3] for the web, Swift[4] for iOS, and Kotlin[5] for Android, to highlight the differences and similarities in syntax and drawing libraries of those three programming languages.

And then I said, why not in other languages? And I started piling them up. At this time there are ~~20~~ ~~21~~ ~~23~~ ~~25~~ ~~26~~ ~~27~~ ~~29~~ 33 of them.

In each project I try to use the default package manager, patterns and best practices. For example ~~Conan~~[6] vcpkg[7] for C++ and C, pip[8] for Python, npm[9] for TypeScript, Composer[10] for PHP, and so on. Modern languages like Go[11], Rust[12], and .NET have a built-in package manager, and that's very handy.

In the cases of Common Lisp[13] and Smalltalk[14] I had to first learn the language from scratch before starting. I used this exploration as the source for my article about Smalltalk in De Programmatica Ipsum[15],

[1] https://gitlab.com/akosma/Conway
[2] https://en.wikipedia.org/wiki/Conway%E2%80%99s_Game_of_Life
[3] https://gitlab.com/akosma/Conway/-/tree/master/TypeScript
[4] https://gitlab.com/akosma/Conway/-/tree/master/Swift
[5] https://gitlab.com/akosma/Conway/-/tree/master/Kotlin
[6] https://conan.io/
[7] https://vcpkg.io/
[8] https://pypi.org/project/pip/
[9] https://www.npmjs.com/
[10] https://getcomposer.org/
[11] https://gitlab.com/akosma/Conway/-/tree/master/Go
[12] https://gitlab.com/akosma/Conway/-/tree/master/Rust
[13] https://gitlab.com/akosma/Conway/-/tree/master/CommonLisp
[14] https://gitlab.com/akosma/Conway/-/tree/master/Smalltalk
[15] https://deprogrammaticaipsum.com/the-absolute-no-frills-quite-ignorant-very-incomplete-and-certainly-flawed-beginners-guide-to-smalltalk/

published in October 2020.

For the Pascal[16] and FreeBASIC[17] versions I had to remember how to write them, as I had not used any of these languages since the mid nineties.

All the implementations share the same very simple architecture: the world of cells is represented as a hash table whose keys are coordinates, and whose values are cells; cells being those things that can be alive or dead at any time. A world can "evolve" into its next stage of life, applying the rules of life to each cell. Most languages provide something akin to a hash table, except for C[18], which took me to add one[19] inspired from some code[20] I found online.

All projects share the same unit tests, too, written in the default unit test library provided by each language. For Objective-C[21], however, I could not find a suitable framework to use, so I just created a separate executable[22] with `assert()` calls in it. I wanted to use an old Objective-C unit test framework called UnitKit[23], written by Duncan Davidson[24] (the creator of Tomcat and Ant, by the way) but couldn't make it work.

Most versions only work on the terminal, drawing the grid in pure text. Only the first three original implementations mentioned above (TypeScript[25], Swift[26], and Kotlin[27]) have GUI representations, but I have not updated the mobile projects in years, and I don't think they can work anymore. The web app in TypeScript still works. Maybe the C++[28] could be used for a GUI app with Qt[29] or wxWidgets[30], maybe. I would love to use the C#[31] code to write a text-based UI using gui.cs[32] by none other than Miguel de Icaza.

The implementation that took me the longer was FreeBASIC[33], simply because I hated it. It's really a horrible language to write code with.

[16] https://gitlab.com/akosma/Conway/-/tree/master/Pascal
[17] https://gitlab.com/akosma/Conway/-/tree/master/FreeBASIC
[18] https://gitlab.com/akosma/Conway/-/tree/master/C
[19] https://gitlab.com/akosma/Conway/-/blob/master/C/src/conway/hashmap.h
[20] https://www.cs.yale.edu/homes/aspnes/pinewiki/C(2f)HashTables.html?highlight=%2528CategoryAlgorithmNotes%2529
[21] https://gitlab.com/akosma/Conway/-/tree/master/Objective-C
[22] https://gitlab.com/akosma/Conway/-/blob/master/Objective-C/tests.m
[23] https://github.com/duncan/unitkit
[24] https://twitter.com/duncan
[25] https://gitlab.com/akosma/Conway/-/tree/master/TypeScript
[26] https://gitlab.com/akosma/Conway/-/tree/master/Swift
[27] https://gitlab.com/akosma/Conway/-/tree/master/Kotlin
[28] https://gitlab.com/akosma/Conway/-/tree/master/C%2B%2B
[29] https://www.qt.io/
[30] https://wxwidgets.org/
[31] https://gitlab.com/akosma/Conway/-/tree/master/C%23
[32] https://github.com/migueldeicaza/gui.cs
[33] https://gitlab.com/akosma/Conway/-/tree/master/FreeBASIC

The ones that I like most are F#[34] and Rust[35]. I just fell in love with these languages.

As another curiosity, I also keep statistics of the lines of code taken by each project, extracted with the cloc[36] tool. It turns out that Pascal[37] and Rust[38] are the longest implementations (around 340 lines) while F#[39], Python[40], and Common Lisp[41] are the shortest (around 140 lines).

I plan to add more languages to the project, but I don't really follow a plan. What I have been doing lately, however, is to update some implementations to the latest versions of each language. For example, I've lately updated projects to Python 3.10[42] and Java 17[43], and this month I'll do the same with PHP 8.1 (enums!) and C# 9. Sometimes these updates decrease dramatically the total number of lines of code of a project.

The project is in GitLab[44], free for everyone to use.

Update, 2022-06-17:: I've added a version using Dart[45].

Update, 2022-07-15: I've added versions in Perl and Crystal[46].

Update, 2022-08-26: I've added a version using the D programming language[47] and another in pure JavaScript, the latter shown in action at the beginning of this article.

Update, 2023-04-14: I've added another using the Zig programming language[48].

Update, 2023-07-14: I've added a version in Minimal BASIC[49], aka ECMA-55[50], compatible with Commodore 64, BBC Micro, AppleSoft BASIC and other flavors. Check it out![51]

[34] https://gitlab.com/akosma/Conway/-/tree/master/F%23
[35] https://gitlab.com/akosma/Conway/-/tree/master/Rust
[36] https://github.com/AlDanial/cloc
[37] https://gitlab.com/akosma/Conway/-/tree/master/Pascal
[38] https://gitlab.com/akosma/Conway/-/tree/master/Rust
[39] https://gitlab.com/akosma/Conway/-/tree/master/F%23
[40] https://gitlab.com/akosma/Conway/-/tree/master/Python
[41] https://gitlab.com/akosma/Conway/-/tree/master/CommonLisp
[42] https://gitlab.com/akosma/Conway/-/commit/3659b3473a4007630ae3c70aafc00c35456e0da1
[43] https://gitlab.com/akosma/Conway/-/commit/247a22cf0a399eda0cf0a594b91f78602e0f7c5b
[44] https://gitlab.com/akosma/Conway
[45] /blog/dart-is-boring/
[46] /blog/crystal-is-a-surprise/
[47] https://dlang.org/
[48] https://ziglang.org/
[49] https://en.wikipedia.org/wiki/Minimal_BASIC
[50] https://www.ecma-international.org/publications-and-standards/standards/ecma-55/
[51] https://gitlab.com/akosma/Conway/-/tree/master/MinimalBASIC

Update, 2023-08-04: I've added versions in QBasic[52] and in VB-Script[53].

Update, 2023-08-18: Added versions in Rexx[54], COBOL[55], Fortran[56], and Visual Basic 1.0[57].

[52] https://gitlab.com/akosma/Conway/-/tree/master/QBasic
[53] https://gitlab.com/akosma/Conway/-/tree/master/VBScript
[54] https://gitlab.com/akosma/Conway/-/tree/master/Rexx
[55] https://gitlab.com/akosma/Conway/-/tree/master/COBOL
[56] https://gitlab.com/akosma/Conway/-/tree/master/Fortran
[57] https://gitlab.com/akosma/Conway/-/tree/master/VisualBasic

Notes About Cloud Without Compromise

Adrian Kosmaczewski

2021-11-12

I've been reading "Cloud Without Compromise: Hybrid Cloud for the Enterprise"[1] by Paul Zikopoulos and Christopher Bienko. I think it's a brilliant book; concise, easy to read, funny, and a great resource for non-technical people who would like to understand how the modern "Cloud" works. Highly recommended.

Here are some notes I took.

Mantra

The book repeats this mantra over and over again:

> "Cloud is a capability, not a destination."

Selected Quotes

About the evolution of terminology and technology:

> When was the last time you heard someone at work say, "I found it on our intranet?" Answer: you were probably dancing the Macarena and weren't embarrassed doing it.

Page 12

Remember to turn off your unused services:

> Our pro tip: always remember that public cloud pricing is utility pricing—the meter is running, just like how you're billed for electricity in your home (and that can be a good or a bad thing as illustrated earlier). Running a nonstop application, or even a developer forgetting to shut down their test instances (one of us, whose name isn't Chris, did that and cost his department $5,000), is like walking around your empty house with all the lights on and wondering aloud

[1] https://www.ibm.com/resources/books/cloud-without-compromise/

about all the money you could be saving. (If you have kids, this is a well-known feeling to you.)

Page 29

Probably the most brilliant quote from the book is this one:

> Every client we talk to is doing one of two things: spending money to save money or spending money to make money. When you spend it (money) to save it, you're renovating and when you spend it to make it, you're innovating.

Page 31

The main goal of all this is speed:

> The theme in modern application development is speed. Speed of scale, speed of delivery, and so on. The very nature of speed is the DevSecOps delivery nuance—bringing together the speed developers love (they think project to project), and the resiliency and assuredness that operations seek (they think about things like service level agreements, upgrades, patching, and security).

Page 44

About Kubernetes:

> We'd be so bold as to say that Kubernetes is the operating system for distributed systems of containers regardless of the hardware architecture. What does an operating system like Linux, Windows, or iOS really do? It manages resources (CPU, memory, storage, communications, and so on) and that's exactly what K8s does: it manages the resources of a cluster of nodes—each of which is running its own operating system.

Page 51

About the Microservices architecture:

> But it's more than that. In a modern system, you might need several databases, an authentication service, a service that manages stock in the warehouse, a service that does billing, a service that queues orders to be shipped, a service that computes taxes, a service that manages currency exchange, and so on. Make your own list for your own business: it will be very extensive. And if your application is used heavily, you may need many copies of these services to handle the load.

Page 52

Putting technical terms in a language that managers can understand is an underrated gift. Paragraphs such as the following are really, really helpful:

> Moving to the cloud gives you increased agility and an opportunity for capacity planning that's similar to the concept of using electricity: it's "metered" based on the usage that you pay for (yes, you'll yell at developers who leave their services running when not in use the same way you do your kids for leaving the lights on in an empty room). In other words, IaaS enables you to consider compute resources as if they are a utility like electricity.

Page 66

The Various Styles of Standup Meetings

Adrian Kosmaczewski

2021-11-19

The most visible star of the Agile galaxy is, without any doubt, the famous, the dreaded, the hyped, the all mighty standup meeting. During the early ages of Agile, from 2005 to 2010, every company I have worked at interpreted the meaning and purpose of the words "standup meeting" according to their own experience, culture, and most importantly, their hierarchy.

In this article I will describe the various styles of standup meetings I've experienced in my career. The words "cargo cult" don't even start to convey the levels of weird I've witnessed in some places.

Oh, and before you ask, yes, all of these stories are true.

The Expensive

In my first job where I experienced the standup meeting around the mid 2000s, they decided that the whole software development department had to participate in such ceremony every morning.

In practical terms, this meant 25 people would sit for an hour and a half in a meeting room, where each person would tell in excruciatingly detail during three minutes what they had done the day before, what they were planning to do that day, and what issues they were facing. Some people would jump in and start criticizing decisions done by others, or even discuss about code or implementation details right in the middle of the meeting.

The problem was that not everyone was working on the same parts of the final product; therefore, most of the discussions were strictly and utterly irrelevant to most of the attendees.

Every day, 25 people earning salaries above 10'000 CHF per month, sitting in a room for 90 minutes. Total estimated cost: around 2'300 CHF per day, or 46'000 CHF per month, or, yes, half a million per year.

Oh, and our product was two years late.

The Noisy

At some other place, we had our standup meeting in our big open space. The problem was that said open space was located next to the cafeteria, and for some reason, somebody had decided to have our standup meeting at precisely the same time all the rest of the company had their “znüni” break (that’s the official name of the 9 AM coffee break in the German side of Switzerland).

A big cacophony ensued, with a dozen engineers screaming at one another to be heard, while the coffee machines were working at full throttle, with around 40 people meeting to chat and laugh.

And to add insult to injury, given the chosen time, of course we missed the coffee break with our colleagues every single day.

To fix this obviously wrong timing choice, the CEO complained to our managers that we were screaming during the “znüni”. Everyone has their own priorities in life.

So our manager decided, mostly to ensure his progression in the ranks of the company, to move the meeting to a closed room. But the office manager had previously decided that this meeting room required a table in the middle. Lo and behold, they requested a table that was so big, we literally had to sit on top of it to make room for everyone to enter the room and close the door. So much for a standup meeting, sitting on top of a big table.

Furniture police later asked us not to sit on that table anymore because we risked scratching it. So we stood next to the door, very awkwardly.

Four months later they removed the table, and we finally had a standup meeting with everyone actually standing up, and without anyone screaming, for the first time.

The Irrelevant

In another job our CEO joined the IT team stadup meeting, and would literally read, from his smartphone calendar app, the agenda of the previous day, providing details of every meeting and every lunch and every single prospect in the CRM.

Then it would swipe on his smartphone with a finger, and repeat the experience, but this time for the current day. All in all, 10 minutes.

The rest of the team, all engineers, would then spend 10 seconds each simply saying whether they had a problem or not.

Which was useless, since all of us were working on different stuff, without the slightest intersection in our duties. There was, simply put, nothing to coordinate.

The Micromanagerial

Since the Agile Manifesto[1] was published, lots of managers all over the world thought the standup meeting was just another great mechanism of fine-grained control.

In such minds, the purpose of a standup meeting is none other than the direct measure of productivity of each employee involved, with little or no interest in the actual ideas behind the word “Agile”. Like, you know, having small autonomous teams that can decide by themselves how to work in the best possible way by collaborating with one another, and stuff like that.

In such situation, our boss would ask us for the reasons behind each gap between commits in the repositories we were working on. Why did your change take a whole afternoon? Why did you create a separate branch? How many tests did you write? Why didn’t you reply to my email before leaving home? Why did you leave so early, by the way? And so on and so forth. This was also sprinkled with the shaming of those who were not being productive enough, according to the narrow views of this manager.

The existence and survival of this simulation of an agile structure prompted this manager to also forbid all remote working, because, well, micromanagement.

More than a “standup meeting” it was a “standup interrogation”; like a “good cop, bad cop” routine, but without the good cop.

The Expeditive

Last but not least, at my team in VSHN[2] we adopted a remote-friendly version of the standup meeting way before the pandemic struck. A quick one-liner on the team chat, saying what’s in our plate for the day, and asking for help if required.

One line, 30 seconds max, every day. It works very well for us. We have a 45 minute-long weekly meeting every week on Monday to coordinate actions, anyway, so such a short standup meeting is more than enough in our opinion.

Other teams in VSHN use other kinds of standup meetings, and I know that some of them have changed the schedule, modality, and requirements of the meeting, according to their wishes.

[1] https://agilemanifesto.org/
[2] https://www.vshn.ch/

Sustainable Ebook Strategy

Adrian Kosmaczewski

2021-11-26

I love reading, and I love books; and in the 21st century, reading books means, to a large degree, electronic books. The advantages are obvious; a small ebook reader is able to hold hundreds or thousands of volumes that would otherwise be impossible to carry around. You can instantly search for text, highlight and bookmark quotes, and a lot more.

But the world of ebooks is a mess; lots of formats, vendors trying to lock-in their customers at all costs, and the eternal issue of DRM. The very idea of DRM on ebooks is an abomination, an abhorrent reality, blocking people from reading books in any way they want.

It took me a while to find out a proper combination of hardware and software to build a sustainable ebook library; that is, one that is based on open standards, able to whitstand the passage of time and the troubled waves of technology. This article summarizes my current setup and why I think it's the best for me. Maybe it works for you too.

Hardware

I have tried at least five different kinds of ebook readers: plain laptops, Kindles, iPads, Android tablets, and finally a Kobo device.

I gave up on Kindles and iPads, as each tied me to their respective ecosystem of online shops and reading experiences. I did not like either. In particular, the iBooks format in the iPad and the Kindle format for ebooks are both the same kind of cancer; something to avoid at all costs. Yes, they offer interesting layout and content options, but they are not standard.

I do not want to be in a position where my reading experience depends on the whims of a company, like Microsoft who decided to remove books from their customers' devices[1] because it got tired of selling books. So using standard formats is a must. I cannot even begin to comprehend how a company can behave like that.

Android tablets are unusable pieces of… technology in general (let's stay polite), so don't even bother. The quality of ebooks reader apps

[1] https://www.npr.org/2019/07/07/739316746/microsoft-closes-the-book-on-its-e-library-erasing-all-user-content?t=1637350342354

in Android also leaves a lot to be desired.

I tried other ebook readers on the iPad, such as Marvin 3[2] by Appstafarian[3] (by far the best I've tested, sadly no longer maintained), or even the long-gone Stanza[4] for iPhone and iPod Touch, but none actually worked for me. iOS devices have very bright screens; that's great for editing pictures, but not so much for reading books. They don't compare to e-ink devices for longer reading sessions, and in particular when reading outdoors in bright daylight.

For the past five years I've used a Kobo[5] Aura H2O device, a fantastic, durable, affordable, simple, and quite powerful device that I use almost every day. It's lightweight, waterproof (it has come with me to the beach and the mountain), and has fantastic battery life, lasting weeks on a single charge. But most importantly, it supports ebooks in EPUB format and without DRM, which was a primary requirement when I bought it.

This Kobo ebook reader even supports one of my preferred services: Pocket[6], which means that I can quickly synchronize my reading list before boarding a train or a plane, and read it comfortably offline.

If I have to buy a new ebook reader in the future (my current one doesn't show any signs of fatigue, though) I will be able to read my collection of books without any problem, thanks to the EPUB standard. And it will probably be a Kobo device, for sure.

Software

I manage my ebook library (~900 titles at the time of this writing) with calibre[7]. It's simply perfect: it's cross-platform, comes bundled with a great EPUB reader, and this Kobo plugin[8] automatically converts standard EPUBs into the Kobo EPUB format, which makes many books faster to scroll and nicer to read.

The Kobo plugin for Calibre can be installed directly from the Preferences > Plugins > Get new plugins menu, and then search for "KoboTouchExtended" by Joel Goguen. Once installed, use the "Send to device" button to automatically convert EPUBs into Kobo EPUBs upon transferring to the reader.

Thanks to this plugin, I can keep my collection in EPUB format in Calibre (future-proof), and I can enjoy it in a more dedicated format (albeit proprietary) in my device. And since Calibre is cross-platform, if

[2] https://apps.apple.com/app/id1086482858

[3] http://appstafarian.com/

[4] http://www.gutenbergnews.org/20080714/stanza-ebook-reader-for-the-iphone-and-ipod-touch/

[5] https://www.kobo.com/

[6] https://getpocket.com/

[7] https://calibre-ebook.com/

[8] https://github.com/jgoguen/calibre-kobo-driver

I ever move to another operating system, my book library will follow me intact.

Other allies in my quest for generating DRM-free EPUBs are Pandoc[9] (great to save a whole website into a single EPUB file) and the Asciidoctor EPUB3[10] module for Asciidoctor[11], which is my preferred way to generate EPUB files.

As for PDFs, I only truly read them on my laptop. I mostly use the PDF reader bundled with Ubuntu, but on the Mac, iOS and Android I prefer using PDF Viewer[12] made by the awesome team of of PSPDFKit[13] assembled by my friend Peter Steinberger[14].

Shopping

As a principle, I do not buy books from Amazon; neither in paper nor ebooks. I use Amazon only to check the catalog and read reviews, and then I buy my books directly at the editors' websites, or, if possible, directly at the writer's website.

I think it is important to support writers directly, so when a writer offers an interesting book on their website, I buy it directly from them. It is important to to cut the chain of intermediate agents between readers and writers as much as possible.

I usually buy or download DRM-free ebooks from the following providers:

- Apress[15]
- Feisty Duck[16]
- Gumroad[17]
- Project Gutenberg[18]
- InformIT[19]
- Leanpub[20]
- Lulu.com[21]
- No Starch Press[22]
- Packt[23]

[9] https://pandoc.org/
[10] https://docs.asciidoctor.org/epub3-converter/latest/
[11] https://asciidoctor.org/
[12] https://pdfviewer.io/
[13] https://pspdfkit.com/
[14] https://twitter.com/steipete/
[15] https://apress.com/
[16] https://www.feistyduck.com/
[17] https://gumroad.com/
[18] https://www.gutenberg.org/
[19] https://informit.com
[20] https://leanpub.com/
[21] https://lulu.com
[22] https://nostarch.com/
[23] https://packtpub.com/

- Pragmatic Bookshelf[24]
- Ray Wenderlich[25]
- Springer[26]
- Standard Ebooks[27]

Most of them offer ebooks without DRM, which is perfect. I want to be able to read those books in the devices that I will buy years from now.

As a side note, Standard Ebooks[28] is a great way to read timeless classics without DRM and with excellent quality; you should check it out, and please consider making a donation if you can.

About DRM and Other Formats

I occasionally do, however, buy books from some non-Amazon bookstores with an interesting catalog online, even though they sell ebooks protected with DRM. In those cases, I remove the DRM after download, so that I can read them without restrictions in any device I want.

Of course, I do not share my DRM-freed ebooks in any way; this procedure is just for my personal consumption. If you can, you should buy the books, and support the writers and their work. And thankfully, more and more publishers are dropping DRM from their ebooks these days.

I'm not going to explain how to remove the DRM from your ebooks; there are plenty of locations online where you can learn that, the same way I did. Please do not message me asking me for advice on this matter either. I will not reply to you.

As for the ebooks I had in the Apple iBooks format, I consider them lost. They are incompatible with anything else, and thankfully I only had two of them, anyway. I do not plan on repeating this same mistake again.

[24] https://pragprog.com/
[25] https://www.raywenderlich.com/whats-new
[26] https://springer.com
[27] https://standardebooks.org/
[28] https://standardebooks.org/

Managing Professional Decline as a Developer

Adrian Kosmaczewski

2021-12-03

My friend Gabriel Garcia Marengo[1] shared a great article by Arthur Brooks[2] a few weeks ago, and it prompted me to reflect on how developers cope with age. This is not a new subject[3] for me; but it is one that, as I approach the glorious age of 50, becomes more and more present in my thoughts.

As Mr. Brooks says in his article and video[4], professional decline is unavoidable. In the software development world, technology moves so fast that the usual recommendation of specializing[5] in a single technology, is a double-edged sword. Specializing is a great idea in the short term, but can be a terrible curse in the long run.

I, for one, have chosen to embrace being a generalist. I love knowing a bit about everything, and that has certainly helped me in my career, which can be roughly sketched out as a continuous hyperjump from one technology galaxy to another[6], following the market and the opportunities therein.

The first large-scale technology keynote I watched online was the introduction of .NET by Bill Gates[7] in June 2000. Watching that keynote, and then installing the first beta of Visual Studio.NET in August that year, it dawned on me that my career as a "VBScript developer" was over. Here was Microsoft telling everybody, "this is the new place we're going", and although I had not seen such a thing before, I could feel the winds of change and the rug being pulled under my feet.

Lo and behold, 2 years later I found my first job as a C# developer. In the meantime I taught myself as much as I could about .NET, went to Microsoft events, installed every single Visual Studio.NET version I could, and my career moved on.

---

[1] https://www.linkedin.com/in/gabrielgm/

[2] https://www.theatlantic.com/magazine/archive/2019/07/work-peak-professional-decline/590650/

[3] /blog/being-a-developer-after-40/

[4] https://www.youtube.com/watch?v=WU2XajpRdgM

[5] https://philipmorganconsulting.com/positioning-manual/

[6] /blog/the-developer-guide-to-migrate-across-galaxies/

[7] https://www.zulenet.com/see/BillGatesNET.html

But I must thank my curiosity once again; because just as I was working with C# and .NET, I bought an Apple iBook G3, and started learning about Objective-C in the evenings (I know, I didn't have much of a social life back then. I give you that.)

And by 2007, when the iPhone emerged as the hottest technology in town, and Microsoft was drowning under the incompetence of Steve Ballmer, I started a business making iPhone (and later Android) apps full-time.

I stayed in this galaxy for 10 years, until 2018. But during that time I also watched the emergence of Docker and Kubernetes; mostly because many iOS projects I worked on required a small backend, too, so keeping up with "server-side" technologies was a must.

And, when I moved away from the iOS and Android galaxy… I ended up in the Docker and Kubernetes one.

In a way, I did specialize, but only for smaller amounts of time. I was a VBScript specialist for 4 years. I was a .NET specialist for 8 years. I was a mobile app specialist for 10 years. And now I've been a working exclusively with Docker & Kubernetes for almost 3 years.

These days I can rely on all of this baggage of 25 years of experience and follow what Arthur C. Brooks said in the article above; I reached a point where I can teach and "connect the dots" across galaxies, find similarities and divergences. De Programmatica Ipsum[8] is, to a large degree, an expression of that situation in life; not only mine, but also that of my friend and co-author Graham[9].

(Speaking about Graham, he dives in such "dot connecting" experiences in his Twitch channels DosAmigans[10] and ObjCRetain[11] together with Steven Baker[12]. Check them out.)

These days I am familiar and fluent in various programming languages[13] at once; not that I'm an expert in any of them, but I can recognize their mutual advantages and suitability for different problems. I have witnessed in person quite a few historical events[14] in our industry. I have been even able to smell the evolution of technology[15] a few years in advance.

And, to be honest, it's no miracle of me; it's actually because all of it is just a little bit of history repeating[16]. Reading about the history of computing has actual, tangible, benefits.

[8] https://deprogrammaticaipsum.com/
[9] https://www.sicpers.info/about/
[10] https://www.twitch.tv/dosamigans
[11] https://www.twitch.tv/objcretain
[12] https://twitter.com/srbaker
[13] /blog/poliglot-conway/
[14] /blog/steve-jobs-vs-bill-gates-on-stage/
[15] /blog/cocoa-is-the-new-carbon-the-future-of-apples-beloved-framework/
[16] https://deprogrammaticaipsum.com/history-repeating/

Being a generalist is like being able to finish a decathlon decently and survive the experience, instead of being a 100 meter olympic champion and subsequently retire at the age of 30.

Some people want the gold medals on their shelves. I prefer to look back and be happy that I was there, and that whatever happened, it happened, and it was good.

Docker and Kubernetes have already both reached the point of commodity, and they are at the peak of their popularity right now. It is going to go down, inexorably, and will be replaced by other things. This is just me preparing for the “next big thing”, learning other new things and being curious as always.

And reading these words you can see the thing I’m actually specializing in: developer relations and communications. Whatever technology life throws at me, there’s always going to be people interested in learning about it, and that’s a lifetime opportunity.

Tech moves on, but people stay.

Languages

Adrian Kosmaczewski

2021-12-10

I have the immense chance and privilege of being fluent in three beautiful languages such as Spanish, French, and English. I have studied and worked with all three, and through them I have come to discover incredible cultures and fantastic works of art.

In all of those discoveries, I realized that I preferred each of them in a specific medium:

- Spanish is my preferred language for **prose**. I find it colorful, rich in adjectives and nouns of various origins, with beautiful verb forms, and a unique sonority. The substance with which Jorge Luis Borges[1] and Julio Cortázar[2] built the worlds that they gave us to dream in.
- In my eyes, French is the language of **poetry**. A poem in French has a strength and a beauty that is unparalleled, a dream-like flight at dusk in a far away planet. It is the matter of tears, shivers, thrills, Charles Baudelaire[3], Guillaume Apollinaire[4], Serge Gainsbourg[5]…
- Finally, for me English is the language of **music**. Songs in English simply sound better. This one is close to the previous, I agree, but it has to do with the natural sonority of the language; the sentence structure is terse and rich, allowing singers to say a lot with few words, and with lots of rhyme and play. I'll just mention Lee Hazlewood[6], Nick Drake[7], Rumer[8], Beck[9], Sia[10], and Madeleine Peyroux[11] along a list of singer/songwriters that would never be complete.

Of course you might agree or not, but that's not the point. Beauty is in the eye and the ears of the beholder.

[1] https://en.wikipedia.org/wiki/Jorge_Luis_Borges
[2] https://en.wikipedia.org/wiki/Julio_Cort%C3%A1zar
[3] https://en.wikipedia.org/wiki/Charles_Baudelaire
[4] https://en.wikipedia.org/wiki/Guillaume_Apollinaire
[5] https://en.wikipedia.org/wiki/Serge_Gainsbourg
[6] https://en.wikipedia.org/wiki/Lee_Hazlewood
[7] https://en.wikipedia.org/wiki/Nick_Drake
[8] https://en.wikipedia.org/wiki/Rumer_(musician)
[9] https://en.wikipedia.org/wiki/Beck
[10] https://en.wikipedia.org/wiki/Sia_(musician)
[11] https://en.wikipedia.org/wiki/Madeleine_Peyroux

Memories of Centralized SCMs

Adrian Kosmaczewski

2021-12-17

It might sound incredible to younger developers out there, but there was a time when Git[1] did not exist. In retrospect, the fact that Git has reigned supreme in its category for over 15 years was previously unheard of. SCM systems came and went in a steady succession since the early 1980s.

Nowadays, Git has become the de facto standard tool; it is hard to remember a time when, during the onboarding of a developer, we had to learn the ins and outs of yet another Source Code Management (SCM) system. "This is how you checkout the project, this is how you commit a change, have fun."

In my first decade as a software developer, I worked with at least five different SCM tools before I even used Git for the first time: Rational ClearCase[2], Microsoft Visual SourceSafe[3], Microsoft Team Foundation Server[4], CVS[5], and Subversion[6]. That's a new SCM tool every two years in average. All of them were centralized systems.

Yet, since 2008, I have seen only a handful of teams not using Git. The SCM wars have ended long ago, and Git has won. Yes, some teams are still using other tools; for example, I did work with a lonely team using Mercurial[7], but it was simply because it was written with Python, and for some teams that's a good enough excuse to adopt a tool instead of another.

Git has won, and in consequence we have switched the conversation to a higher level of abstraction. Teams debate about using either GitHub[8], GitLab[9], Gitea[10], or even Bitbucket[11] for their GitOps[12]

[1] https://git-scm.com/
[2] https://en.wikipedia.org/wiki/Rational_ClearCase
[3] https://en.wikipedia.org/wiki/Microsoft_Visual_SourceSafe
[4] https://en.wikipedia.org/wiki/Azure_DevOps_Server
[5] https://en.wikipedia.org/wiki/Concurrent_Versions_System
[6] https://en.wikipedia.org/wiki/Apache_Subversion
[7] https://en.wikipedia.org/wiki/Mercurial
[8] https://github.com/
[9] https://about.gitlab.com/
[10] https://gitea.com/
[11] https://bitbucket.org/
[12] https://www.gitops.tech/

needs. Because, yes, don't say DevOps out loud, that's soooooo 2009.

(By the way, Bitbucket now conveniently advertises itself as a "Git solution for professional teams" when they actually started as an alternative to GitHub based on Mercurial.)

Git's triumph brought some self-enforcing stabilization in the world of IDEs. Gone are the days when Mac OS X 10.2 "Jaguar"[13] developers used Project Builder[14], an IDE featuring... native CVS integration. Xcode, the heir of Project Builder born in 2003, only included native Subversion support in version 3.0 (that's around 2009, iPhone OS 2.1, anyone?); it later added a native Git client during the 2010s, and now even features GitHub and GitLab integrations.

Git is so convenient and so prevalent these days that Visual Studio Code[15] supports it off-the-box. Of course, you can add GitLens[16] on top of it, but I don't think it's such a good idea. There's plenty of useful desktop Git clients, anyway: Tower[17], GitKraken[18], TortoiseGit[19], Sourcetree[20], Magit[21], fugitive.vim[22]...

Then in 2011 Vincent Driessen invented git-flow[23] and the Internet went bezerk with people screaming at each other how they were doing Git branching wrong[24]. Because dogmas are like that.

What people don't remember, and I actually think it's a good thing, is how it felt working with non-distributed, centralized SCM systems like Visual SourceSafe or Subversion. Because, well, it was a sport.

We have to understand that businesses in particular were very afraid of conflicts caused by two developers editing the same file. Back in the 90's, "enterprise" SCMs allowed developers to "lock" a file in the server, so that nobody else but them could edit it, at all. Can you imagine that? I know, it's hard to picture, so here's a sequence of commands of the `scm-tool`, a ficticious... SCM tool that never made it big in this market.

```
$ scm-tool lock scm://server/project/file.cpp
$ vim project/file.cpp
$ scm-tool commit file.cpp -m "Changed things"
$ scm-tool unlock scm://server/project/file.cpp
```

[13] https://en.wikipedia.org/wiki/Mac_OS_X_Jaguar
[14] https://en.wikipedia.org/wiki/Project_Builder
[15] https://code.visualstudio.com/
[16] https://www.gitkraken.com/gitlens
[17] https://www.git-tower.com/mac
[18] https://www.gitkraken.com/
[19] https://tortoisegit.org/
[20] https://www.sourcetreeapp.com/
[21] https://magit.vc/
[22] https://github.com/tpope/vim-fugitive
[23] https://nvie.com/posts/a-successful-git-branching-model/
[24] https://deprogrammaticaipsum.com/you-are-doing-it-wrong/

Having seen this, here's a Gedankenexperiment: what happens if the last command in the snippet above is forgotten? And even worse, what if this happens on a Friday evening before the developer leaves for their well-deserved, long-dreamt two-month trip to the Kerguelen Islands[25]? You guessed it! Nobody, and I say nobody in the team, will be able to edit that file until said colleague returns. Well, maybe the sysadmin could unlock it with some sudo magic, but what if this person is also in the Kerguelen Islands?

Ah, the 1990s. Well, should the scenario above happen, I guess we would have watched yet another episode of Friends[26], and shrugged it all off. After all, the Agile Manifesto[27] had not been written yet, so there was no need to feel bad if we weren't releasing software at every heartbeat.

And think about the benefits: no more conflicts! You would never have to drill down and fix those lines starting with <<<<<<< in your source code ever again! A beautiful side effect.

The thing that we did lose forever with distributed SCMs was, thankfully, database corruptions and its malevolent twin, the complete project history loss. Yes, when you had a centralized server, particularly like in the case of that miserable failure of a system that was Microsoft Visual SourceSafe, you ran the risk of losing your entire project history because of the lack of atomic transactions. And, unlike Git, you did not clone projects, but you merely checked out the last version; which means that the whole history of the project was at risk at every single commit.

Visual SourceSafe was so prone to these failures that systems like Rational ClearCase, Subversion, and SourceGear Vault[28] based their marketing around the fact that their commits, unlike their Microsoft brethren, were actually transactionally safe. How about that.

The world of SCMs was such a uncontrolled mess of conflicting options and crappy software, that Joel Spolsky explicitly stated that using it was the number one step for better code[29]. Most teams just didn't use an SCM system at all. At my first job, we didn't use one. What a time to be alive.

[25] https://en.wikipedia.org/wiki/Kerguelen_Islands

[26] https://en.wikipedia.org/wiki/Friends

[27] https://agilemanifesto.org/

[28] https://www.sourcegear.com/vault/

[29] https://www.joelonsoftware.com/2000/08/09/the-joel-test-12-steps-to-better-code/

Three Key Tenets of GNOME App UX

Adrian Kosmaczewski

2021-12-24

If you are developing apps for GNOME[1], I kindly ask you to pay attention to the following three key points of application usability. These are features that all GNOME GUI apps should support no matter what. In the humble opinion of this author, of course.

- CTRL+Q is the primary mechanism to quit your app. Not ALT+F4, please, that's a Windows thing, and a carpal-tunnel syndrome-inducing keystroke at best.
- F11 must toggle the application back and forth from full screen mode.
- CTRL+M should minimize the app.

And that's it. You can map the rest of the keyboard any way you want, but please pay attention to these three items. The good thing is that many, if not most apps do support these keyboard shortcuts off-the-box.

[1] https://www.gnome.org/

The Wrong Question

Adrian Kosmaczewski

2021-12-31

I once had a second interview for a job at a small software engineering company. Instead of the common programming questions I was expecting, the person who interviewed me asked the following: “how much do you think a developer costs me?”

I was quite surprised by the question, and for a few seconds I simply didn’t know what to answer. I replied that I had run my own business for a while, so I knew a bit about all the ancillary costs of having an employee in Switzerland: insurances, pension funds, etc., on top of a regular salary. But I was still frowning, trying to gauge the actual intention behind the question.

I guess my answer pleased this person; in retrospect I should have left the room right there and then. I took the job because I needed it, and I only lasted there a few months.

As expected, after a few weeks in the job this manager became convinced that I was working less than the 42 hours and 30 minutes of legal work per week.

Because, you see, in Switzerland the law states eight hours and a half per day of working time, or 42.5 hours per week; most companies just specify eight per day (40 per week) in their standard contracts, though.

Actually this was the first time in my life in Switzerland where I had to work 42.5 hours per week. And it was also the first time I had only 20 days of legal leave per year, which again, is the minimum length specified in said Swiss law. Most companies give you 25 days, and some even more[1], because, you know, markets, competition, and such.

Here was a business who liked to squeeze as much work as possible out of its workforce, by offering the strict minimum in exchange.

(The longer I keep writing this article, the more I ask myself how on Earth I thought it was a good idea to sign that contract, or to stay in that job for that long. Anyway.)

[1] https://www.vshn.ch/en/blog/vshn-introduces-additional-vacation-days-for-each-year-of-service/

So as I was saying, they thought that I was "stealing" hours from them. Now, the thing is that I did work those hours. I did not work less time, and I'd say that I actually worked more than that: usually I would start on the morning train, and finish on the train back home. Since I had 40 minutes of train in each direction, I used the time there to start or finish some tasks. Add 4 hours in the morning and 4 in the afternoon, that makes for 8 hours and 40 minutes of net working time. Heck, sometimes I'd even work on evenings. I'm such an idiot.

But no, their idea of work was for me to be physically present in the office all those eight and a half hours a day. (Kids, this was before the pandemic, when working from home was unheard of, unless you worked at Basecamp[2], that is.)

And boy did they get angry. I actually got yelled at several occasions, and even got threatened of legal action. I replied to such courtesy by asking what proof they had that I was not completing my hours. I even had the toupet of suggesting them to install one of those time tracking machines with badges, so that they could still accuse people of misconduct and felony if they wanted, but, you know, with actual proofs.

And they re-yelled at me, that they did not have the obligation of installing such machines (indeed Swiss law does not say anything about that, point taken), but that I had the obligation of working the legal amount of hours (again, touché).

Clearly it didn't matter to them that all of my projects were on time, or that the customers I was dealing with were very pleased with my output, or that I had contributed substantially in quantity and quality, in every project I touched. Or, you know, that it was their word against mine: I did work the required 42.5 hours a week, and more.

The only concern of this business was for me to work the maximum number of hours allowed by law in situ, and having the least possible amount of holidays allowed by said law, and to pay the least possible salary that would pass the market test. That was it.

All of this while, at the same time, they were telling everyone that this company was "a family" (more on that in another article[3]) and that if we did not attend some of the various company BBQs they set up arbitrarily following their whims and desires, we would have one of those 20 days of legal leave discounted, because retaliation and punishment are great ways to nurture that family feeling.

One day I resigned, and in retrospect, it was one of the best decisions I took.

A few months later I received an invitation from them to one of those BBQs, because they like to keep this "family" thing going on with their alumni. And it's a large alumni family all right, particularly judging by

[2] https://basecamp.com/books/remote

[3] /blog/we-are-family/

the length of the output of the `git shortlog -s -n -e` command in each repository. Spoiler: that list was longer than the list of current employees. If that isn't a red flag, I don't know what is.

Said invitation ended up in the recycling bin, of course, but fate wanted me to meet one of those managers in a train station a few weeks later. He rushed to me like a kid, asking if I had received the invitation, and became furious when I told him that yes, I had gotten it, and that no, thanks but no thanks. His face actually turned red. I guess he wanted to sue me or something right there and then.

I turned around, wide eyed and chuckling, and walked away.

So, long story short: if you ever get asked about how much you will cost a company where you would like to work, instead of being asked how much value you can bring thanks to your knowledge, just walk away. The only thing some businesses care about is the P&L sheet, where salaries (and for extension, employees) are just liabilities, and you don't want to work for such "families".

We Are Family

Adrian Kosmaczewski

2022-01-07

Here is one of the most blatant lies I've heard in my professional career, and sadly, I've heard it at quite a few jobs: "We are a family". For some reason, to compensate for otherwise abysmally dystopian work environments, founders and managers fall into the temptation of telling their staff that they "are like family" to them, or that the whole company "is like one big family".

This is wrong, at (oh so) many levels. It is actually a huge red flag in my book of self-preservation.

Let's say it clearly: your workplace and your family are two different things. As much as you can (and should) enjoy being with great colleagues (and I truly hope you do), your job and your family are two different things.

(Well, unless you work in your own family's "family business", of course; in which case, well, your colleagues have a high probability of actually being your family, from a genetically point of view. If that's the case, you might also happen to have shares in said business, or to benefit from it substantially in various ways, which is quite a bonus point, but entirely beyond the scope of this article.)

Most of us don't work in a family business, and most of us don't even own voting shares of the companies where we happen to work. In those cases, work is… work. It is, in essence, just a contractual relationship that binds you to certain obligations in exchange for certain rights, the most visible and widespread of which is, of course, a regular pecuniary salary.

That's it. There's nothing else. And none of the definition above sounds like "family". If it does, well, I'm nobody to judge. The hard truth is: you can be fired from work in case of misconduct, bankruptcy, or simply because they decide so! One day you show up for work, a cup of latte in your hands, and your boss breaks the news that your position has been terminated. Been there, had that.

So much for a family, if you ask me.

Work is a mutually convenient exchange of workforce for cash and other perks. Both parties take out something good from the exchange, and are better off afterwards, so that everybody is happy about doing

it; which is why we do it. We both win something out of this exchange. There might be some added value on top of that: do you get bonuses at the end of the quarter? A company car? Massages? Free food? A foosball table? It is all well and good, but it's not that important at the end of the day.

Thing is, the most important parts of work are those that do not appear in the job description. Things like the friendliness of the people you work with. The trust they have vested in you and your capacity of decision, and of course in the output of your work. The truth you have for them. The respect everyone has for one another at every single moment they're together. The level of collaboration, solidarity, and support you get from your peers when shit hits the fan.

And I want to stress this last point: you will realize whether you're at the right place (or not) as soon as the first problem appears. Tough situations are the ones that actually make everyone show their true nature.

Every single place where I've heard the phrase "we are all family here" from a Michael Scott[1] in charge, was, inexorably, a toxic hornet nest where a few assholes made everyone else feel miserable every single moment of their lives. A place of burnout, people crying in the bathrooms, political fights for infinitesimal quotas of power, discrimination against everyone that is not a thirty year-old white caucasian male, and worse.

And there's one more hidden aspect behind that phrase. Many people in our modern world are physically and emotionally alone; the reasons are various and I won't delve in them; but it's a fact of life. Saying "we are family" is just speculating on top of that loneliness, providing a seemingly safe haven in a context that isn't one. It's a way of manipulating people to work uncompensated overtime and to "give it all" in exchange for love. It's a way to create a tribe based on the fear of losing the love of a family that isn't. It's sinister, demeaning, abject, and it happens in a lot of workplaces.

So, if you ever hear the phrase "we are family", follow my advice, and run away for dear life.

[1] https://www.imdb.com/title/tt0386676/

The Argentine Brain Drain

Adrian Kosmaczewski

2022-01-14

Argentina is currently experiencing a brain drain so strong that it made the headlines on Swiss television[1]. 200 Argies, many with higher education degrees or quite a bit of professional experience, are leaving the country… every day. To put that in context, that's around 6'000 per month, or 70'000 per year.

As terrible and damaging as this sounds, this is, sadly, not an unprecedented situation.

Abroad

The last Nobel Prize laureate in sciences awarded to an Argentine[2] (medicine 1984, by Dr. César Milstein[3]) was awarded to an Argie living abroad. There's been no other Argentine Nobel Prize ever since.

Two of the greatest writers in Argentine literature lived and passed away abroad (Jorge Luis Borges[4] in Geneva, Julio Cortázar[5] in Paris) after suffering censorship, condemn, and in some cases, actual threats to their life and work.

Buried in the same cemetery[6] as Borges, Alberto Ginastera[7] is widely considered one of the greatest musicians of the 20th century; his opera Bomarzo[8] was censored[9] by the dictatorship of Juan Carlos Onganía[10]. Martha Argerich[11], the best pianist in the world as I write these lines, has lived and worked outside of Argentina for the past 60 years.

[1] https://www.rts.ch/info/monde/12766669-largentine-enregistre-une-fuite-des-cerveaux-sans-precedent.html

[2] https://en.wikipedia.org/wiki/List_of_Nobel_laureates_by_country#Argentina

[3] https://en.wikipedia.org/wiki/C%C3%A9sar_Milstein

[4] https://en.wikipedia.org/wiki/Jorge_Luis_Borges

[5] https://en.wikipedia.org/wiki/Julio_Cort%C3%A1zar

[6] https://en.wikipedia.org/wiki/Cimeti%C3%A8re_des_Rois

[7] https://en.wikipedia.org/wiki/Alberto_Ginastera

[8] https://en.wikipedia.org/wiki/Bomarzo_(opera)

[9] https://es.wikipedia.org/wiki/Juan_Carlos_Ongan%C3%ADa#Censura_art%C3%ADstica

[10] https://en.wikipedia.org/wiki/Juan_Carlos_Ongan%C3%ADa

[11] https://en.wikipedia.org/wiki/Martha_Argerich

Past

For more than half a century, Argentina has tried as hard as it could to push intellectuals out of the country, committing a slow institutionalized suicide. This almost imperceptible rotting process is at the root of the decadence of what was considered one of the richest and most advanced countries at the beginning of the 20th century.

And to give an example of this national suicide, let's go back to the aforementioned Juan Carlos Onganía for a bit. In July 29th, 1966, just a month after having overturned a democratic government, the de facto government of Onganía (dubbed the "Argentine Revolution"[12]) decided, following the guidelines of the dictatorship workbook, that thinking people was a threat to his power. Hence, "The Night of the Long Batons"[13] happened.

Why did Onganía physically attack professors and students, destroyed labs, burned libraries, and single-handedly ended most of Argentina's scientific progress on just one sweeping move? Because said students and professors were defending the university reform of 1918[14], which stated the required autonomy for universities to carry out secular education and research, free from government and religious intervention.

And, you see, that meant "communist" ideas might slip into the minds of people. God forbid sentient beings being free to discuss and analyze ideas. The very catholic General Onganía felt a divine duty to stop such madness and bring the country back to the right track.

As a result of this sadistic attack on the University of Buenos Aires, 300 professors, 200 of which were researchers, left the country and, in many cases, never returned.

This is the case of Manuel Sadosky[15], a mathematician and pioneer of computer science in Latin America, who built and run the first computer in the country: "Clementina", a Ferranti Mercury[16] computer. His daughter Cora Sadosky[17], who also left the country, gave her name to the Sadosky Prize[18], given every two years by the Association for Women in Mathematics[19] to an outstanding young female researcher in maths.

(As an anecdote, and strangely enough, Dr. Sadosky was a childhood friend of Cortázar.)

This was not the last suicidal self-harm attack on Argentine education institutions and members thereof; suffice to mention the abominable

[12] https://en.wikipedia.org/wiki/Argentine_Revolution
[13] https://en.wikipedia.org/wiki/La_Noche_de_los_Bastones_Largos
[14] https://en.wikipedia.org/wiki/Argentine_university_reform_of_1918
[15] https://en.wikipedia.org/wiki/Manuel_Sadosky
[16] https://en.wikipedia.org/wiki/Ferranti_Mercury
[17] https://en.wikipedia.org/wiki/Cora_Sadosky
[18] https://en.wikipedia.org/wiki/Sadosky_Prize
[19] https://awm-math.org/

"Night of the Pencils"[20] of 1976, perpetrated by yet another military government, the "National Reorganization Process"[21], still convinced in their delusion, as always, of their divine duty to rid their country from the threat of communism.

Present

Nowadays, the reasons for those 200 argentines to leave the country every day are not political. They are not being persecuted or attacked. They simply cannot cope with rampant insecurity, lack of research infrastructure, explosive inflation, an outstandingly stupid and regressive tax system, a ridiculously complex bureaucracy system, a decaying health apparatus, and more. I understand them[22].

Actually, now that I think about it, of the 30 people in my high school class in 1990, I know of at least 10 who moved to another country in the past 30 years. Some have returned (I have, for a short while, 1998-2001) but most are still abroad.

Around 22 years ago I tried to teach a friend from Argentina some basic concepts of computer science. Among those was that of a "process"; precisely the one concept explained in the first page of the first chapter of "Structure and Interpretation of Computer Programs, Second Edition"[23] by Abelson, Sussman, and Sussman:

> We are about the study the idea of a computational process. Computational processes are abstract beings that inhabit computers.

As soon as I said the word "process", he looked at me in disgust, mentioned the "National Reorganization Process"[24] of 1976, and walked away. His family and friends had been persecuted, merely 20 years before our meeting, and some of them had disappeared, never to be found again.

There's still a great deal of pain in the country around what happened in the last 50 years. Everything is very fresh, to the point where people are still being threatened and effectively kidnapped and assassinated[25] by far right groups.

Future

Speaking about economic development (and, as it happens, computers in general) in such an environment is simply not possible. The

[20] https://en.wikipedia.org/wiki/Night_of_the_Pencils

[21] https://en.wikipedia.org/wiki/National_Reorganization_Process

[22] /blog/thirty-years/

[23] https://mitpress.mit.edu/9780262510875/structure-and-interpretation-of-computer-programs/

[24] https://en.wikipedia.org/wiki/National_Reorganization_Process

[25] https://en.wikipedia.org/wiki/Disappearance_of_Jorge_Julio_L%C3%B3pez

steps in the “Maslow pyramid of needs” of a society must be taken gradually, from bottom to top, and such conditions are not in place at the moment, and haven’t been for half a century.

And for sure, it is not the classical neoliberal recipe of taking more debt, reducing government spending, or another of those classic ideas floating around economic circles that will bring Argentina back on its feet.

The key to doing that consists in stopping everything, dropping the “macho, tango, & football” attitude, asking for forgiveness for all the destruction caused, and understanding that one cannot build a country just with crops and excellent meat. It takes education, and with it, the development of culture, literature, cinema, all of the arts, together with science and technology, to build a nation. Those are the major pillars, the ones that generate wealth in the long run.

There are countries who have rebuilt themselves from scratch. It can be done. But it takes 30+ years, and thus, a commitment to having at least eight (8) successive democratically chosen governments sharing the same vision, with continuity, strength, wisdom, and faith.

But this is, precisely, the hardest part for a country like Argentina.

Text Editors for Work

Adrian Kosmaczewski

2022-01-21

There has been a particular text editor that defined each period of my career as a software developer. This article is a summary of that history, so far.

I'm not mentioning the various IDEs that I had to use in the same points in time, because they are really specific to a particular technology: Visual Studio for C#, Xcode for Objective-C and Swift, etc. This is only about small, fast, flexible tools that have helped me write prose and code throughout the years.

If you like this, I've written more about the subject in one of my books[1] (EPUB[2] and HTML[3] version).

EditPlus (1998–2007)

I loved EditPlus[4] since the first day I opened it, and it was my personal companion at writing code for years. Small, incredibly fast, with plenty of syntax files[5] contributed by users. I used it to write all of my HTML, CSS, Active Server Pages, VBScript code, and more with it. As I stopped working on Windows, I stopped using it. I'm glad it is still around after all these years, though.

Smultron (2003–2006)

I bought an iBook G3 in December 2002 to learn Unix and Objective-C; but for a couple of months I struggled to find a text editor as good as EditPlus. The first one that actually worked for me was Smultron[6], a rather obscure text editor from Sweden, by far not the most popular in the platform, but extremely good at what it did.

[1] /books/Tales_Of_Editors_And_Keyboards/Tales_Of_Editors_And_Keyboards.pdf
[2] /books/Tales_Of_Editors_And_Keyboards/Tales_Of_Editors_And_Keyboards.epub
[3] /books/Tales_Of_Editors_And_Keyboards/Tales_Of_Editors_And_Keyboards.html
[4] https://editplus.com/
[5] https://editplus.com/files.html
[6] https://www.peterborgapps.com/smultron/

TextMate (2006–2011)

TextMate[7] was the editor that David Heinemeier Hansson used to demo Ruby on Rails. We can still watch this demo in what is now one of the oldest videos in YouTube[8]. The success of Ruby on Rails propulsed TextMate to the top; however, the lack of progress towards TextMate 2.0 made me look elsewhere.

MacVim (2011-2016)

I used MacVim[9] for a few years on the Mac, as my default GUI text editor. It had pretty good integration with MacOS, and it used the same shortcuts and plugins as the standard vim, so it was perfect in every way.

vim (2002–now)

As I said, one of the reasons I bought my iBook G3 in 2002 was to learn the Unix command line. And that's how I started using vim[10] every day. I started piling up customizations in my `.vimrc`, then started using the Janus[11] distribution, and finally settled for just a few selected plugins installed with Pathogen[12]. Almost 20 years after, I think learning vim was one of the best investments of time. I use it in every computer I work with.

Visual Studio Code (2016–now)

I started using Visual Studio Code[13] in 2018, as fate took me to work in a few projects based on TypeScript. I loved it, and these days I use it for anything that does not happen on a command line. I've written an article about the myriad of extensions[14] that I use with it. It has great support for pretty much anything that I do, and it is cross-platform, which means that as I moved to Linux[15], it came with me.

Emacs (2018-now)

I had never used Emacs[16] before 2018, and I wrote a book[17] as I learnt to use it, mostly comparing it with vim. I like Emacs a lot, both in GUI

[7] https://macromates.com/
[8] https://www.youtube.com/watch?v=Gzj723LkRJY
[9] https://github.com/macvim-dev/macvim
[10] https://www.vim.org/
[11] https://github.com/carlhuda/janus
[12] https://github.com/tpope/vim-pathogen
[13] https://code.visualstudio.com/
[14] /blog/lots-of-vscode-extensions/
[15] /blog/migrating-from-macos-to-linux/
[16] https://www.gnu.org/software/emacs/
[17] /books/Tales_Of_Editors_And_Keyboards/Tales_Of_Editors_And_Keyboards.pdf

and TUI mode, and in particular I appreciate its integration with Git using Magit[18] and its Asciidoc mode[19].

Update, 2022-11-11: I can't believe I left MarsEdit[20] out of this article. Of course I didn't use it to write code, but writing blog posts was part of my work for a long time, and I wrote plenty of them with this application.

[18] https://magit.vc/

[19] https://www.emacswiki.org/emacs/AsciiDoc

[20] https://redsweater.com/marsedit/

Open Letter to SaaS Apps

Adrian Kosmaczewski

2022-01-28

Dear SaaS app developer, entrepreneur, marketer, CEO, etc; here's a tip: don't annoy users, particularly those who have paid accounts.

It is not uncommon for your paying users to own and use several different devices. They will want to use your app (for which they gladly pay a monthly or yearly subscription) on their personal laptop, on a tablet, and then in their mobile. Maybe even in a separate desktop computer, too. They will want to log in each and all of those devices using their personal account, and will want to enjoy the benefits and services you offer.

What I'm asking is that, when they do so, do not bother them by logging them out from the other devices. Let them use more than one device at a time (DeepL, I'm looking at you) or more than two (SurveyMonkey, I see you.) Do not log them out from a previous device they've used to access your application, or block them from logging to a new one. It's obnoxious, seriously.

May I remind you that even a paid Microsoft 365 subscription can be used in up to five (5) devices at once, simultaneously. Heck, even Evernote can be used in up to two (2) devices in their free-as-in-beer entry-level plan.

Are you really going to log me out? Seriously? Every single time? Yes, I do switch devices often. Sometimes I even use different browsers in the same device. Because reasons.

So when you log me (a paying user) out of a previous device when I use another one, I feel irritated. Do not log me out. Let me use your service. I'm a paying user. I want to support you. Don't log me out.

This is particularly valid if you're an independent SaaS, not yet bought up by or otherwise associated with big tech. Don't annoy your users. Let them use up to 5 devices. It's not that complicated. Just 5.

I don't pretend to use 100 devices with your service. Just 5 at most.

Shut up and take my money. Well, yes, until I get tired of your shit and I don't want to give it to you anymore.

Thanks.

Epic Trailer Songs

Adrian Kosmaczewski

2022-02-04

Have you noticed how many movie trailers lately feature classic songs (from the 60's, 70's, 80's, and 90's) remastered in some kind of "Epic" mode? Take a look:

- "Enjoy the Silence"[1] by Depeche Mode (released in 1989) in the "Ghost in the Shell"[2] (2017) trailer[3].
- "Hurt"[4] from Trent Reznor / Nine Inch Nails (released in 1995), covered by Johnny Cash in the "Logan"[5] (2017) trailer[6].
- "Gangsta's Paradise"[7] from Coolio (released in 1995) in the "Valerian and the City of a Thousand Planets"[8] (2017) trailer[9]
  - Although another trailer[10] used "Because"[11] from the Beatles, released in 1969.
- "Take On Me"[12] (released in 1985) from A-ha in the "Ready Player One"[13] (2018) trailer[14]
  - There's another trailer[15] that mixes "World in my Eyes"[16] by Depeche Mode, released in 1990, with "Jump"[17] by Van Halen, released in 1984.
- "Mony Mony"[18] (released in 1968), cover by Billy Idol in the "Hellboy"[19] (2019) trailer[20].

[1] https://en.wikipedia.org/wiki/Enjoy_the_Silence
[2] https://www.imdb.com/title/tt1219827/
[3] https://www.youtube.com/watch?v=G4VmJcZR0Yg
[4] https://en.wikipedia.org/wiki/Hurt_(Nine_Inch_Nails_song)
[5] https://www.imdb.com/title/tt3315342/
[6] https://www.youtube.com/watch?v=Div0iP65aZo
[7] https://en.wikipedia.org/wiki/Gangsta\XeTeXglyph\numexpr\XeTeXcharglyph"0027\relax{}s_Paradise
[8] https://www.imdb.com/title/tt2239822/
[9] https://www.youtube.com/watch?v=K8oVfkZM3pA
[10] https://www.youtube.com/watch?v=cPeqNTqZNN0
[11] https://open.spotify.com/track/1rxoyGj1QuPoVi8fOft1Kt?si=e805a18c47eb495d
[12] https://en.wikipedia.org/wiki/Take_On_Me
[13] https://www.imdb.com/title/tt1677720/?ref_=nv_sr_srsg_0
[14] https://www.youtube.com/watch?v=rjLVCpE3kuw
[15] https://www.youtube.com/watch?v=cSp1dM2Vj48
[16] https://open.spotify.com/track/0TBn49AjkNufCRJ2O5VJ6s?si=0c0595bb7ceb46b6
[17] https://open.spotify.com/track/6Fba9RZtC6vTY814JToDtP?si=10793d95b38843b4
[18] https://en.wikipedia.org/wiki/Mony_Mony
[19] https://www.imdb.com/title/tt2274648/
[20] https://www.youtube.com/watch?v=dt5g5_1cKVk

- "Blue Monday"[21] by New Order (released in 1983) in the "Wonder Woman 1984"[22] (2020) trailer[23].
- "Eclipse"[24] of Pink Floyd (released in 1973), remixed by Hans Zimmer in the "Dune"[25] (2021) trailer[26].
- "Sweet Dreams"[27] by Eurythmics (released in 1983) in one trailer[28], and Blondie's "Heart of Glass"[29] (released in 1978) in another[30] of the "House of Gucci"[31] (2021) trailers.
- "White Rabbit"[32] by Jefferson Airplane (released in 1967) in the "Matrix Resurrections"[33] (2021) trailer[34].
- "Notorious"[35] by Duran Duran (released in 1986) in the "Red Notice"[36] (2021) trailer[37].
- "Something in the Way"[38] by Nirvana (released in 1991) in "The Batman"[39] (2022) trailer[40].

Interestingly, some of these movies (with some exceptions, of course, for example "Logan", or "Ready Player One") had abysmal reviews and rating. In those cases, the trailer was the best part of the movie. Sad yet interesting trend.

Update, 2022-04-22: "Sweet Child of Mine"[41] by Guns N' Roses (released in 1988) in the "Thor: Love and Thunder"[42] (2022) trailer[43].

Update, 2022-10-07: "Clocks"[44] by Coldplay (released in 2002) in the "Slumberland"[45] (2022) trailer[46].

---

[21] https://en.wikipedia.org/wiki/Blue_Monday_(New_Order_song)
[22] https://www.imdb.com/title/tt7126948/
[23] https://www.youtube.com/watch?v=sfM7_JLk-84
[24] https://en.wikipedia.org/wiki/Eclipse_(Pink_Floyd_song)
[25] https://www.imdb.com/title/tt1160419/?ref_=nv_sr_srsg_0
[26] https://www.youtube.com/watch?v=n9xhJrPXop4
[27] https://en.wikipedia.org/wiki/Sweet_Dreams_(Are_Made_of_This)
[28] https://www.youtube.com/watch?v=eGNnpVKxV6s
[29] https://open.spotify.com/track/41AE8ODX2JETBCa6muhSLY?si=3ebd7ad895a64c87
[30] https://www.youtube.com/watch?v=pGi3Bgn7U5U
[31] https://www.imdb.com/title/tt11214590/?ref_=fn_al_tt_1
[32] https://en.wikipedia.org/wiki/White_Rabbit_(song)
[33] https://www.imdb.com/title/tt10838180/
[34] https://www.youtube.com/watch?v=9ix7TUGVYIo
[35] https://en.wikipedia.org/wiki/Notorious_(Duran_Duran_song)
[36] https://www.imdb.com/title/tt7991608/
[37] https://www.youtube.com/watch?v=T6l3mM7AWew
[38] https://en.wikipedia.org/wiki/Something_in_the_Way
[39] https://www.imdb.com/title/tt1877830/?ref_=nv_sr_srsg_0
[40] https://www.youtube.com/watch?v=mqqft2x_Aa4
[41] https://en.wikipedia.org/wiki/Sweet_Child_o\XeTeXglyph\numexpr\XeTeXcharglyph"0027\relax{}_Mine
[42] https://www.imdb.com/title/tt10648342/
[43] https://www.youtube.com/watch?v=BZCBEw6GeYE
[44] https://en.wikipedia.org/wiki/Clocks_(song)
[45] https://www.imdb.com/title/tt13320662/
[46] https://www.youtube.com/watch?v=FBnkVJslRGo

Update, 2022-10-24: “Goodbye Yellow Brick Road”[47] by Elton John (released in 1973) in the “Ant-Man and the Wasp: Quantumania”[48] (2022) trailer[49].

Update, 2022-11-29: “Fame”[50] by David Bowie (released in 1975) in the Babylon[51] (2022) trailer[52]

[47] https://en.wikipedia.org/wiki/Goodbye_Yellow_Brick_Road_(song)
[48] https://en.wikipedia.org/wiki/Ant-Man_and_the_Wasp%3A_Quantumania
[49] https://www.youtube.com/watch?v=ZlNFpri-Y40
[50] https://en.wikipedia.org/wiki/Fame_(David_Bowie_song)
[51] https://www.imdb.com/title/tt10640346/
[52] https://www.youtube.com/watch?v=5muQK7CuFtY

Things That Define Big Software Companies

Adrian Kosmaczewski

2022-02-11

Looking at the software industry, it appears that most big companies usually share more traits than they would like to admit. Take for example their products: at any given time, big software companies all had at least one product of various similar categories, roughly grouped in three big areas.

The companies fitting such a description are the usual suspects:

- Microsoft
- Apple
- Google
- Amazon
- Oracle
- IBM

Let's start with the foundational parts first.

- **Computers (duh!).** All of these companies sell their own hardware things running their own software inside. Yes, Amazon sells Kindles, IBM sells mainframes[1], and Oracle sells SPARC servers[2].
- **Operating systems.** Windows, z/OS[3], macOS, Solaris[4], Chrome OS[5], Amazon Linux[6], you name it. If you're a big company, you gotta have your own operating system, or a Linux distro with some added value to sell for big bucks to enterprise customers. Usually POSIX[7]-compatible, whatever that is. You can even have many; nowadays Microsoft has two Linux distros of their own (one for Azure[8], another for WSL[9]), and in the 1980s they had their own Unix[10]. How about that. By the way, the

[1] https://www.ibm.com/it-infrastructure/z
[2] https://www.oracle.com/servers/
[3] https://en.wikipedia.org/wiki/Z/OS
[4] https://en.wikipedia.org/wiki/Oracle_Solaris
[5] https://en.wikipedia.org/wiki/Chrome_OS
[6] https://aws.amazon.com/amazon-linux-2/
[7] https://en.wikipedia.org/wiki/POSIX
[8] https://github.com/microsoft/CBL-Mariner
[9] https://github.com/WhitewaterFoundry/Pengwin
[10] https://en.wikipedia.org/wiki/Xenix

rush for a “modern” operating system is exactly what almost made Apple bankrupt[11] in the 1990s.

- **Office suites;** more or less compatible with the formats used by the biggest of all of them, Microsoft Office, even if lately the OpenDocument[12] standard is making inroads. Interestingly, this format family originally debuted in what became Oracle’s own StarOffice[13]. Lately these office suites been transpiled to JavaScript or even more lately Wasm[14], like Google Docs, Office 365, and Apple iWork, for example. They all suck and developers went back to pure text, usually in the form of Markdown[15] or Asciidoc[16] to get actual work done. And if all else fails, there’s PDF.
- **Web browsers;** previously, companies would actually work on their own HTML & JavaScript engines, but these days even Microsoft uses Chromium. New browsers are great until you use them for real. I stick with Firefox or LibreWolf[17], thankyousomuch. Even Amazon have (had?) their own browser[18] and lots of products with the “Fire” moniker, fitting many of the categories above and below. Even IBM used to have its own web browser[19] for another of its operating systems, that marvel of the 90’s called OS/2[20]. Le sigh.

The second area of interest are developer tools.

- **Cloud platforms or “IaaS”;** everybody has their own these days, starting with the[21] big[22] three[23], and behind them Oracle[24] and IBM[25]. As for Apple’s iCloud[26], more an SaaS than anything, it offers some shy PaaS features[27] for developers to use in mobile apps. With one billion users, it’s hard not to count iCloud in this list.
- **Programming languages;** always proprietary, more or less cross-platform, and lately very open source: C#[28], Swift[29], Go[30],

[11] https://www.wired.com/1997/06/apple-3/
[12] https://en.wikipedia.org/wiki/OpenDocument
[13] https://en.wikipedia.org/wiki/StarOffice
[14] https://en.wikipedia.org/wiki/WebAssembly
[15] https://daringfireball.net/projects/markdown/
[16] https://asciidoctor.org/
[17] https://librewolf.net/
[18] https://docs.aws.amazon.com/silk/
[19] https://en.wikipedia.org/wiki/IBM_WebExplorer
[20] https://en.wikipedia.org/wiki/OS/2
[21] https://aws.amazon.com/
[22] https://azure.microsoft.com/en-us/
[23] https://cloud.google.com/
[24] https://en.wikipedia.org/wiki/Oracle_Cloud
[25] https://en.wikipedia.org/wiki/IBM_Cloud
[26] https://en.wikipedia.org/wiki/ICloud
[27] https://developer.apple.com/icloud/cloudkit/
[28] https://dot.net/
[29] https://www.swift.org/
[30] https://go.dev/

Java[31], COBOL[32]. To gain the heart of all developers, they are all multi-paradigm with both lambdas and objects, so you can do functional and OOP all in the same app. They can all (sorta) do desktop, backend, frontend, cloud, mobile, and Hello World. They are all bloated in their own special ways and people love and hate them in various proportions. And Amazon supports them all[33].

- **IDEs,** usually serving as a support and glue for all the other items in this section. Visual Studio remains a strong contender, and will celebrate its 25th birthday on March 17th. On the other side, Apple should phase out that bad joke of Xcode, and instead ask somebody who actually knows a thing or two about IDEs to help, like Google did when it asked JetBrains[34] to create a new Android Studio. We can mention Eclipse, which serves as the basis of whatever IBM offers[35], and NetBeans, which serves Oracle[36] well. And Amazon has Cloud9[37].
- **Self-sufficient backend programming systems:** COBOL, ASP.NET, Amazon Lambda, J2EE... Even Apple had one, a long time ago, called WebObjects[38]. If they wanted they could make a bigger PaaS out of iCloud, and use Swift all over their datacenters. They could; if they don't, it's not because a lack of cash. On the other hand, this is a domain where Google particularly shines: they started with GWT[39] and later begat Kubernetes[40], the platform to rule them all. They also moved to the frontend with Angular[41], by the way. Even more: their work on the V8 JavaScript engine[42] gave origin to Node.js[43], arguably the most important web technology of the 2010s.
- **Database engines.** That was mostly a 1990s race, though; these days there's too many too good open source database engines out there, in the FOSS arena, to justify the craziness of coming up with a new one. But for a while, around 1990, there was not a lot of choice and the war was raging. DB/2[44], SQL Server[45], lots of Amazon purpose-built[46] databases, just like Google's[47],

[31] https://java.com/en/
[32] https://developer.ibm.com/languages/cobol/
[33] https://aws.amazon.com/developer/
[34] https://www.jetbrains.com/
[35] https://ibm.github.io/mainframe-downloads/eclipse-tools.html
[36] https://www.oracle.com/tools/technologies/netbeans-ide.html
[37] https://aws.amazon.com/cloud9/
[38] https://en.wikipedia.org/wiki/WebObjects
[39] http://www.gwtproject.org/
[40] https://kubernetes.io/
[41] https://en.wikipedia.org/wiki/Angular_(web_framework)
[42] https://en.wikipedia.org/wiki/V8_(JavaScript_engine)
[43] https://en.wikipedia.org/wiki/Node.js
[44] https://en.wikipedia.org/wiki/IBM_Db2_Family
[45] https://en.wikipedia.org/wiki/Microsoft_SQL_Server
[46] https://aws.amazon.com/products/databases/
[47] https://cloud.google.com/products/databases/

and the eponymous RDBMS that made Larry Ellison[48] a billionaire. Even Apple has something that looks like a PaaS database[49]. And let's also not forget that FileMaker[50] is made by a subsidiary of Apple.

Finally, we have the consumer space. Only Microsoft, Apple, Amazon, and Google are active in this area. I don't expect to see an Oracle smartwatch or an IBM music store anytime soon, if you see what I mean, but I've often been wrong in the past.

- **Game consoles:** that was the craze at the end of the 1990s, when Apple thought the Pippin[51] was a good idea. Actually the iPhone and Android became much bigger mobile gaming platforms than what their creators thought they would be. Gaming is a category where Microsoft actually did something relevant[52] and quite ground breaking.
- **Smartphone platforms;** that's a 2000s fashion, and one category that Microsoft lost completely. Now it's over, Apple and Google are a de facto duopoly now. Amazon wanted to be in this category too, didn't work, in spite of all the Fire.
- **Retail and payments.** The big thing these days. Many of these "software" companies are into retail of mostly digital goods (usually apps, games, books, and music), with the obvious exception of Amazon, who also sells and ships physical goods. And if you sell stuff, you might as well bypass payment companies and offer your own solution there, too, like Apple[53] and Google[54], based on the strong support of their smartphone businesses.

Upcoming battlefields:

- No Code / Low Code platforms (Amazon[55], Google[56] and Microsoft[57] are already present in that space)
- Quantum computing (IBM is investing heavily in it)
- Wearable computing (Apple and Google are there)
- Healthcare, fitness, wellness (Apple is investing a lot in the field)

Here's the gist: software companies became big because all of these bricks generate a positive feedback effect as they reinforce one another, orbiting around the needs of somewhat orthogonal groups of users.

Playing the **platform game** is an expensive, but hugely lucrative game in the long run. Yes, it is expensive; to reach a billion users, you

[48] https://en.wikipedia.org/wiki/Larry_Ellison
[49] https://developer.apple.com/icloud/cloudkit/
[50] https://en.wikipedia.org/wiki/FileMaker
[51] https://en.wikipedia.org/wiki/Apple_Pippin
[52] https://en.wikipedia.org/wiki/Xbox
[53] https://en.wikipedia.org/wiki/Apple_Pay
[54] https://en.wikipedia.org/wiki/Google_Pay
[55] https://www.honeycode.aws/
[56] https://cloud.google.com/appsheet
[57] https://powerapps.microsoft.com/en-us/

have to literally spend tens of billions of your favorite non-devalued currency.

Hewlett-Packard, Xerox, and countless other technology companies could have been part of this group, had they embraced software as an industry and a potential, and not as an afterthought. Facebook could count but they are more of a media company than anything else.

Maybe the Alibaba Group[58] or the Tata Group[59] could follow these steps? Maybe they already do to a large degree.

On the other hand, here's things people thought all big software companies could/should/had to do, but outsiders did them much better:

- Chat.
- Search engines.
- Social media. Apple[60] and Google[61], I'm looking at both of you.

[58] https://en.wikipedia.org/wiki/Alibaba_Group
[59] https://en.wikipedia.org/wiki/Tata_Group
[60] https://en.wikipedia.org/wiki/ITunes_Ping
[61] https://en.wikipedia.org/wiki/Google%2B

Technology Wars

Adrian Kosmaczewski

2022-02-18

There's been plenty of technology wars in the software industry. Here's a list of those that I got to witness in the past 25 years.

- **Methodologies:** in the 90s there was a new software project management methodology every 10 days. Scrum (Agile in general) won.
- **Web browsers:** First there was Netscape from 1993 to 1997, and then Internet Explorer took over the chrown for almost 15 years. Since then it's Chrome all over the way, and even Microsoft ditched all efforts to compete in that market. Chrome won.
- **Source Code Management:** I've talked about this already[1], and Git won. If you don't know what Git is, this might help[2].
- **Text Editors:** TextEdit, UltraEdit, or Notepad++? Vim or Emacs? Joe or Nano? TextMate or Sublime Text? I've also written about this[3]. Never mind, Visual Studio Code won.
- **Spreadsheets:** Lotus 123, Borland Quattro Pro, Lotus Improv remember? Bah, Excel won.
- **Word processors:** again. Remember WordPress, WordStar, Ami Pro, StarOffice / OpenOffice / LibreOffice? Word won.
- **Operating systems:** Windows won. At least on the desktop. One of those things that define[4] big software companies.
- **Container orchestration systems:** Apache Mesos and Docker Swarm were really short lived. Kubernetes won. Again, if you don't know what it is, you're welcome[5].
- **Smartphones:** iOS and Android killed BlackBerry and Windows Mobile, but then again Android won with the numbers.
- **Programming languages:** That's a hard one; COBOL won but then PL/I won and then Pascal won but C++ won and then Smalltalk could have won but Java won the enterprise and then Ruby won the Web 2.0 and then Python won the Machine Learning. Well, according to Atwood's Law, JavaScript won.

[1] https://akos.ma/blog/memories-of-centralized-scms/
[2] https://akos.ma/blog/git-for-non-technical-readers/
[3] https://akos.ma/blog/text-editors-for-work/
[4] https://akos.ma/blog/things-that-define-big-software-companies/
[5] https://akos.ma/blog/kubernetes-for-non-technical-readers/

- **Web servers:** NGINX won. Because, seriously, Apache running on a container? Please.
- **Relational Databases:** Oracle, SQL Server, MySQL... bah, PostgreSQL won. It is too good to be true. Even though SQLite is also quite a winner.

There were the DEC Wars[6] too, and even the Unix Wars[7], but those are different things.

[6] https://www.bsd.org/decwars.html
[7] http://www.catb.org/~esr/writings/unixwars.html

Elixir and Phoenix Framework

Adrian Kosmaczewski

2022-02-25

I've been learning a bit about the Elixir[1] programming language lately, and for that I created a small app using the Phoenix Framework[2].

I had never used Elixir before, so this post contains some information about how I got started. You can see the results of this experiment on GitLab[3].

Install Erlang, Elixir, and Phoenix

Elixir is a language that compiles to BEAM bytecode, running in the Erlang[4] virtual machine. Getting Elixir installed is explained in detail in the Elixir documentation. Here's what I had to do on Ubuntu 20.04:

```
$ wget https://packages.erlang-solutions.com/erlang-solutions_2.0_all.deb
$ sudo dpkg -i erlang-solutions_2.0_all.deb
$ sudo apt update
$ sudo apt-get install esl-erlang
$ sudo apt install elixir
```

The Elixir language itself feels very Ruby-like at first approach, and Phoenix is quite a bit like Rails, but of course with all the power of the Erlang VM behind. All of this is very intentional, of course.

```
defmodule FortuneElixirWeb.PageController do
  use FortuneElixirWeb, :controller

  def index(conn, _params) do
    number = :rand.uniform(1000)
    message = elem(System.cmd("fortune", []), 0)
    hostname = elem(System.cmd("hostname", []), 0)
    version = "1.2-elixir"
    fortune = %{"number" => number, "message" => message, "hostname" => hostnam
    accept = case get_req_header(conn, "accept") do
      [version] -> version
      _ -> "text/html"
```

[1] https://elixir-lang.org/
[2] https://phoenixframework.org/
[3] https://gitlab.com/vshn/applications/fortune-elixir
[4] https://www.erlang.org/

```
    end
    case accept do
      "application/json" -> json(conn, fortune)
      "text/plain" -> text(conn, "Fortune #{version} cookie of the day ##{numbe
      _ -> render(conn, "index.html", fortune: fortune)
    end
  end
end
```

(Yes, it's the same app I made with Rust and Python[5] a while ago.)

Speaking about the framework, here it goes:

```
$ mix archive.install hex phx_new
```

Create and Run a Project

And once you have the framework installed, creating a new project is very simple:

```
$ mix phx.new fortune_elixir --no-ecto
```

The `--no-ecto` option removes the boilerplate for PostgreSQL connectivity, which is otherwise installed by default. I wanted to create the simplest possible project.

Finally, start the project and open it in your browser:

```
$ cd fortune_elixir
$ mix phx.server
```

Connect to http://localhost:4000 to see the application in action.

[5] /blog/first-web-app-in-rust/

Ukraine

Adrian Kosmaczewski

2022-03-04

It has been hard for me to think about anything else than the Russian invasion of Ukraine since last week; it is also hard for me to write about it.

I have friends in Ukraine, and I'm worried about them. It's as simple as that.

I visited Ukraine only once, in July 2016, when I was invited to speak[1] at the UMT conference in Dnipro[2]. We took a high-speed train from Kyiv to Dnipro with Anastasiia[3], Paul[4], and many more.

From the train I saw the infinite yellow fields meeting the light blue sky; I discovered that the Ukrainian flag is actually a photograph.

UMT 2016 was the first conference I ever participated where my family name fit naturally among all the other speakers. I did feel bad because I could not greet people, not even in Polish (I learnt that Polish and Ukrainian were closer than I ever imagined). I did feel, however, very welcome and surrounded by warm and funny people.

I remember trying to decypher the Ukrainian alphabet, in Cyrillic script, figuring out things like food names in menus; and discovering in the airport that Zürich is spelled "Цюрiх".

I am worried about my friends. I am worried about a whole country. But I am proud to know them in person. I am proud to know they're fighting back.

Today I feel Ukrainian.

---

[1] /blog/refactoring-ios-projects/
[2] https://en.wikipedia.org/wiki/Dnipro
[3] https://twitter.com/vixentael
[4] https://twitter.com/TT_Kilew


@TT_Kiev

Gitea

Adrian Kosmaczewski

2022-03-11

GitHub, BitBucket, GitLab; they are not the only solutions available to share Git repos with your friends and colleagues. I've been playing with Gitea[1] recently, and it's a really cool alternative, written in Go[2]. Here's a quick way to test it.

Install

You can run the binary[3] or the container[4], of course, but it's also very easy to install Gitea in Minikube[5]:

```
$ minikube start --addons=ingress
$ helm repo add gitea-charts https://dl.gitea.io/charts/
$ helm install gitea gitea-charts/gitea
```

Create and apply a quick ingress for it:

```yaml
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: gitea-ingress
  labels:
    app: gitea-ingress
  annotations:
    nginx.ingress.kubernetes.io/rewrite-target: /
    kubernetes.io/ingress.class: nginx
spec:
  rules:
    - http:
        paths:
          - path: /
            pathType: Prefix
            backend:
              service:
                name: gitea-http
```

[1] https://gitea.com/
[2] https://go.dev/
[3] https://docs.gitea.io/en-us/install-from-binary/
[4] https://docs.gitea.io/en-us/install-with-docker/
[5] https://docs.gitea.io/en-us/install-on-kubernetes/

```
            port:
              number: 3000
```

After install, proceed as follows:

1. Login as `gitea_admin` (see below) and create a new user.
2. Check the IP of the server using `$ minikube ip`
3. Edit `/etc/hosts` with the line `192.168.49.2 git.example.com` where the IP is the one from above

Here's the default credentials:

- Username: `gitea_admin`
- The default password is in the helm chart[6]: `r8sA8CPHD9!bt6d`

One can also configure Gitea to use LDAP and other authentication mechanisms.

Maybe the biggest drawback of Gitea is that it does not feature a built-in CI/CD pipeline mechanims. There are some options listed here[7], for example Drone[8], Agola[9], or Woodpecker[10].

If you're into OpenShift, here's what you need[11]. And here's a tutorial[12] showing how to use GitOps with Gitea and Argo CD on OpenShift.

[6] https://gitea.com/gitea/helm-chart/src/branch/master/values.yaml
[7] https://gitea.com/gitea/awesome-gitea#user-content-devops
[8] https://docs.drone.io/server/provider/gitea/
[9] https://agola.io/
[10] https://woodpecker.laszlo.cloud/
[11] https://github.com/wkulhanek/docker-openshift-gitea
[12] https://redhat-scholars.github.io/outer-loop-guide/outer-loop/5.1/gitops-workflow.html

BASIC Standards

Adrian Kosmaczewski

2022-03-18

I've learnt lately that there are many standards[1] for the BASIC programming language. Who woulda thunk?

First there's the ECMA-55 Minimal BASIC standard[2], released in 1978. I found an interpreter[3] for it on GitHub. There's also a fully fledged compiler[4], whose author also wrote a book[5] and some interesting slides[6] about it. Talk about passion; the whole idea of this is to bring back the simplicity and excitement of a bare bones programming language, a minimalist environment for people to learn programming in its simplest possible form.

Then there's the ECMA-116 Full BASIC standard[7] of 1987, with its associated implementation[8] ready to be downloaded and used.

The ECMA standards were dropped in favour of the ISO Full BASIC standard ISO/IEC 10279[9] of 1992, confirmed in 2008, and available for a fee from the ISO.

There were a gazillion implementations of the language (Microsoft alone published lots of them[10] in the past 45 years), yet the industry did somehow agree on a few standards; that was a surprise to me.

David Lien wrote the encyclopedia[11] of the language in 1978, covering over 200 implementations for the various personal computers of the time; I would love to find a copy of this book.

---

[1] https://en.wikipedia.org/wiki/BASIC#Standards

[2] http://www.ecma-international.org/publications/files/ECMA-ST-WITHDRAWN/ECMA-55,%201st%20Edition,%20January%201978.pdf

[3] https://github.com/jorgicor/bas55

[4] https://buraphakit.sourceforge.io/BASIC.shtml

[5] https://buraphakit.sourceforge.io/Learn_BASIC.pdf

[6] https://buraphakit.sourceforge.io/ResurrectMinimalBASIC.pdf

[7] http://www.ecma-international.org/publications/files/ECMA-ST-WITHDRAWN/ECMA-116,%201st%20edition,%20June%201986.pdf

[8] http://hp.vector.co.jp/authors/VA008683/english/

[9] https://www.iso.org/standard/18321.html

[10] https://en.wikipedia.org/wiki/Microsoft_BASIC

[11] https://openlibrary.org/books/OL3789525M/The_basic_handbook

Efficient Meetings

Adrian Kosmaczewski

2022-03-25

In this post you will find a collection of interesting book quotes about meetings, and how to make them suck less.

Meetings are Good™ ® ©

Some authors have a positive view on meetings in general.

"The Mythical Man-Month" by Frederick P. Brooks (1975)

(Link[1]) Page 75:

> Regular project meetings, with one team after another giving technical briefings, are invaluable. Hundreds of minor misunderstandings get smoked out this way.

"Agile!: The Good, the Hype and the Ugly" by Bertrand Meyer (2014)

(Link[2]) Page 153:

> **Short daily meetings** focused on simple verbal reports to progress – the "three questions" – are an excellent idea. It need not be practiced in a dogmatic way, since distributed projects and companies with flexible work schedules must adapt the basic scheme, but is one of the practices that undeniably help software development, and deserves to be adopted even more widely than it already is.

Meetings Suck™ ® ©

Some authors, a large number I must say, don't share the same opinion.

[1] https://en.wikipedia.org/wiki/The_Mythical_Man-Month
[2] https://www.amazon.com/Agile-Good-Hype-Bertrand-Meyer/dp/3319051547

“Rework” by David Heinemeier Hansson and Jason Fried (2010)

(Link[3]) Page 108: “Meetings are toxic”

Very, very, very negative outlook. If you need meetings at all, follow these simple steps:

- Set a timer. When it rings, meeting’s over. Period.
- Invite as few people as possible.
- Always have a clear agenda.
- Begin with a specific problem.
- Meet at the site of the problem instead of a conference room. Point to real things and suggest real changes.
- End with a solution and make someone responsible for implementing it.

“More Effective Agile” by Steve McConnell (2019)

(Link[4]) Page 112:

> For general meetings, it’s useful to provide guidance in conducting meetings effectively. At a minimum this should include standard advice: have a clear purpose for the meeting, set clearly defined expectations about what decision or other deliverable the meeting will produce, err on the side of scheduling the meeting to be shorter rathert than longer, invite only people who are necessary to support the meeting’s deliverable, declare the meeting to be over as soon as it has met its objective, and so on. A good resource in this area is How to Make Meetings Work (Doyle, 1993).

“Peopleware: Productive Projects and Teams (3rd Edition)” by Tom DeMarco and Tim Lister (2013, first edition 1987)

(Link[5]) Chapter 31 is all about meetings (page 187)

> A meeting that is specifically called to get something done might be called working meeting, typically called to reach a decision. Who should be invited? That’s easy, the people who need to agree before the decision can be judged made. Nobody else. To make sure no one is blindsided, it’s essential that the working meeting have an agenda relevant to its purpose and that it stick to that agenda.
>
> No one has to attend defensively.
>
> The cost of the meeting is directly proportional to the number attending.

[3] https://basecamp.com/books/rework

[4] https://moreeffectiveagile.com/

[5] https://www.amazon.com/Peopleware-Productive-Projects-Teams-3rd/dp/0321934113

“Leading a Software Development Team: A developer’s guide to successfully leading people & projects” by Richard Whitehead (2001)

(Link[6]) Chapter 5 is all about meetings (page 34). Valid reasons to call a meeting:

- To get people together to talk about a specific topic
- Get a decision by consensus instead of decree
- Discuss something contentious
- When people have not met before, when they start working together, coming from different environments.

Even if it sounds obvious: if a meeting is not necessary, don’t call for one. If someone isn’t needed, don’t call them.

If a meeting has more than about 8 people in it, it’s unlikely that everyone will contribute. That’s fine if it’s a “get everyone together to tell them all something” type of meeting.

Meetings 2h long or more, and senior people are involved, it might make sense to arrange to meet off-site. This gets people away from interruptions, to clear the mind of day-to-day concerns. Excellent for forward-thinking strategy or radical proposals.

Meetings cost a great deal of money. Not only because people’s time is expensive, but because they can’t do their normal jobs if they’re in a meeting.

Chairmanship

1. Before the meeting is held, always send round an agenda. Does not need to be long, just a bullet point per discussion item; make sure people arrive at the meeting having had a chance to think about the issues to be discussed.
2. Wait for everyone to arrive
3. Remind everyone why the meeting takes place.
4. Make sure someone takes notes (or do it yourself.)
5. Keep the meeting on course, sticking to the agenda. Lead the conversation towards the decision that needs to be made.
6. Draw discussions to a close by summarizing.
7. Thank people for coming.
8. Send minutes afterwards.

“Principles” by Ray Dalio (2017)

(Link[7]) Section 4.4 is about meetings (pages 364-368)

[6] https://www.amazon.com/Leading-Software-Development-Team-successfully/dp/0201675269

[7] https://www.principles.com/

> There are many reasons why meetings go poorly, but frequently it is because of a lack of clarity about the topic or the level at which things are being discussed.

1. Make it clear who is directing the meeting and whom it is meant to serve.
2. Be precise in what you're talking about to avoid confusion.
3. Make clear what type of communication you are going to have in light of the objectives and priorities.
4. Lead the discussion by being asssertive and open-minded.
5. Navigate between the different levels of the conversation.
6. Watch out for "topic slip."
7. Enforce the logic of conversations.
8. Be careful not to lose personal responsibility via group decision making.
9. Use the "two-minute rule" to avoid persistent interruptions.
10. Watch out for assertive "fast talkers."
11. Achieve completion in conversations.
12. Leverage your communication.

"The 4-Hour Workweek" by Timothy Ferriss (2007)

(Link[8]) Section about meetings in chapter 7 (page 96): "Interrupting Interruption and the Art of Refusal"

> The third step is to master the art of refusal and avoiding meetings.

1. Steer people toward the following means of communciation, in order of preference: e-mail, phone, and in-person meetings. If someone proposes a meeting, request an e-mail instead and then use the phone as your fallback offer if need be.
2. Meetings should only be held to make decisions about a predefined situation, not to define the problem.

   > Nine times out of ten, a meeting is unnecessary and you can answer the questions, once defined, via e-mail.

3. If you absolutely cannot stop a meeting or call from happening, define the end time. Do not leave these discussions open-ended, and keep them short. If things are well-defined, decisions should not take more than 30 minutes.

"Making Things Happen" by Scott Berkun (2008)

(Link[9]) Chapter 10: "How not to annoy people: process, email, and meetings" (page 205)

> Here is my meeting confession: I do not like regularly scheduled meetings. Unless there is a force keeping them lean

[8] https://fourhourworkweek.com/

[9] https://scottberkun.com/making-things-happen/

> and tidy, they will eventually slide into slow, bloated, frustrating, dysfunctional wastes of time. However, if there is that force in place, meetings can be energizing, centering experiences for everyone in the room.

The art of facilitation Facilitate: the act of making things easy or easier.

> Good meetings happen only when someone in the room understands how to facilitate. (...) Facilitating can be a semi-formal role, held by a designated person who runs the show (often the PM) or by whoever called the meeting.

- Establish a host position
- Listen and reflect
- Direct the conversation
- End the conversation
- Make history

Three kinds of meetings

- **Highly interactive discussion.** Everyone is expected to participate, for depth and intimacy, exploring or resolving specific issues, seeking out alternative ideas. Size: 2-8 people. Examples: brainstorming, design discussion, triage.
- **Reporting or moderate discussion.** One person has content to cover, and she needs people to respond to or to understand that content. Goal to get knowledge. Medium to large (5-15 people.) Examples: small presentation (like this one), spec review.
- **Status and project review.** Summarize the sattus of a team or an entire project, to medium to large groups (10-100 people.) All-hands meeting, project review, status review.

Meeting pointers

- Are the right people in the room?
- Sit or stand
- Prepare
- Laptops and gadgets
- Being on time
- End with clear steps and owners

"Making Ideas Happen: Overcoming the Obstacles Between Vision and Reality" by Scott Belsky (2012)

(Link[10]) Discussion about meetings in page 78.

[10] https://www.amazon.com/Making-Ideas-Happen-Overcoming-Obstacles/dp/1591844118

Most meetings are fruitless. Amidst all the brainstorming, we must find ways to measure the outcome of meetings.

- Don’t meet just because it’s Monday. Abolish automatic meetings without an actionable agenda.
- End with a review of actions captured.
- Call out nonactionable meetings.
- Conduct standing meetings.
- Don’t call meetings out of your own insecurity.
- Don’t stick to round numbers. Instead of 30 or 60 minutes, call for 10 minute meetings instead.
- Always measure with action steps… or something else.

More

I found more information about the subject. There’s a lot out there.

- “The Essence of Technical Communication for Engineers: Writing, Presentation, and Meeting Skills”[11] by Herbert Hirsch, 2000, has a whole chapter about meetings.
- “How to Make Meetings Work”[12] (Doyle, 1993). The title says it all.

Tweets

Finally, an interesting thought on Twitter about the subject.

How to make teams productive:- keep them very small- have as few priorities as you can- do as few hours of meetings as possible

— Jack Altman (@jaltma) July 13, 2020

And a funny one, too.

The key to effective meetings. pic.twitter.com/ZPdnnscCiM

— Northern Soul (@Northern_Soul_) July 14, 2020

[11] https://www.amazon.de/Herbert-Hirsch/dp/0780347382/
[12] https://www.amazon.com/Make-Meetings-Work-Michael-Doyle/dp/0425138704

A Linker for Joel

Adrian Kosmaczewski

2022-04-01

In January 2004 Joel Spolsky wrote a blog post titled "Please Sir May I Have a Linker?"[1], where he described his tribulations trying to install a small .NET[2] app in computers not bundled with the original .NET framework.

18 years later, his wish has been granted. Well, maybe it's been granted a while ago, but I learnt about this only recently.

Here's the thing: one can cross-compile .NET Core 6.0 applications in any platform to any other platform, just like with Go for example. Here's how to create a macOS application (with Intel CPU) on a Linux box:

```
dotnet publish --configuration Release --runtime osx-x64 \
  -property:PublishReadyToRun=true -property:PublishSingleFile=true \
  --self-contained true
```

This command specifically requests the creation of a single file application[3] using ReadyToRun[4], which optimizes its startup time.

The standalone executable will be located in `bin/Release/net6.0/osx-x64/publish` and indeed, fully statically linked, it does not require anything else to run. Copy it to another machine, run, done.

Using my Conway[5] project as an example (there's a C#, a VB.NET, and an F# implementation there) I get a ~60 MB (gasp) application that happily runs without any other requirements.

Well, what Joel asked for included a specific requirement:

> Then, it removes any library functions that your program does not use.

Clearly, if my final executable for such a simple project as this one weighs 60 MB… there's probably a bit of code to remove. Go[6] does this much better than .NET, at least at the time of this writing.

[1] https://www.joelonsoftware.com/2004/01/28/please-sir-may-i-have-a-linker/
[2] http://dot.net/
[3] https://docs.microsoft.com/en-us/dotnet/core/deploying/single-file
[4] https://docs.microsoft.com/en-us/dotnet/core/deploying/ready-to-run
[5] /blog/polyglot-conway/
[6] https://go.dev/

Anyway. Replace the `--runtime` property from `osx-x64` to `win-x64` and you can build a Windows application, whatever operating system you're using:

```
dotnet publish --configuration Release --runtime win-x64 \
  -property:PublishReadyToRun=true -property:PublishSingleFile=true \
  --self-contained true
```

In this case, the standalone executable will be located in `bin/Release/net6.0/win-x64/publish` and again, it does not require any other dependencies.

The complete catalog[7] of runtime IDs is available in the documentation of .NET, but they include pretty much all combinations of Windows, macOS, and Linux with 32 and 64 variants of both Intel and ARM CPU architectures–well, only 64 bits for macOS, of course.

[7] https://docs.microsoft.com/en-us/dotnet/core/rid-catalog

Cross Platform Node.js Apps

Adrian Kosmaczewski

2022-04-08

Node.js projects are very much cross-platform, allowing your team to use any operating system they want to work on them. Well, yes, to a large degree they are; but as always, there are some caveats that can bring some headaches if you have Windows developers in your team.

Environment Variables

When using environment variables like `NODE_ENV` in your `npm` scripts in Windows, the Git Bash command line interpreter in Windows complains that said `NODE_ENV` variable (or any other) "is not a command". To avoid this problem, use cross-env[1] in the application:

```
npm install cross-env —save-dev
```

Now you can use the `cross-env` command before any environment variable you need for your `npm` scripts, and all operating systems will be happy.

Removing Folders

Use rimraf[2]. Do not hardcode `rm -rf` in your `npm` script, because that command won't work on Windows.

```
npm install rimraf —save-dev
```

This is particularly useful for your `npm run clean` command.

[1] https://www.npmjs.com/package/cross-env
[2] https://www.npmjs.com/package/rimraf

Technology Stacks

Adrian Kosmaczewski

2022-04-15

For the past 20 years there's been this trend to give names to a particular set of operating system, programming language, database tech, and something else. Here goes a list with the most common ones.

- LAMP[1] Linux, Apache, MySQL, PHP/Perl/Python
  - XAMPP[2] Apache HTTP Server, MariaDB, PHP, Perl
  - WAMP[3] Windows, Apache, MySQL, and PHP
  - WIMP[4] Windows, IIS, MySQL, and PHP
  - MAMP[5] macOS, Apache, MySQL or MariaDB, PHP, Perl, or Python
  - DAMP[6] Darwin, Apache, MySQL, PHP
  - LEMP[7] Linux, Nginx, MySQL/MongoDB, PHP
  - AMPPS[8] Apache, MySQL, MongoDB, PHP, Perl and Python
- MEAN[9] MongoDB, Express.js, AngularJS (or Angular), and Node.js
- SAFE[10] Saturn, Azure, Fable, Elmish
- ELK[11] Elasticsearch, Logstash, Kibana
- BCHS[12] BSD, C, httpd, SQLite
- LYME[13] Linux, Yaws, Mnesia (or CouchDB), Erlang
- GLASS[14] Gemstone/S, Linux, Apache, Smalltalk, Seaside
- Jamstack[15] static HTML
- GRANDstack[16] GraphQL, React, Apollo, Neo4j

[1] https://en.wikipedia.org/wiki/LAMP_(software_bundle)
[2] https://en.wikipedia.org/wiki/XAMPP
[3] https://en.wikipedia.org/wiki/WAMP_(disambiguation)
[4] https://en.wikipedia.org/wiki/WIMP_(software_bundle)
[5] https://en.wikipedia.org/wiki/MAMP
[6] https://en.wikipedia.org/wiki/DAMP_(software_bundle)
[7] https://lempstack.com/
[8] https://en.wikipedia.org/wiki/AMPPS
[9] https://en.wikipedia.org/wiki/MEAN_(solution_stack)
[10] https://safe-stack.github.io/
[11] https://www.elastic.co/what-is/elk-stack
[12] https://learnbchs.org/
[13] https://en.wikipedia.org/wiki/LYME_(software_bundle)
[14] https://kalountos.wordpress.com/2015/06/03/a-beginners-guide-to-installing-and-configuring-glass-gemstone-64s-smalltalk-seaside-application-server-object-database-server-environment/
[15] https://jamstack.org/
[16] https://grandstack.io/

- XRX[17] XForms, REST and XQuery
- PETAL[18] Phoenix, Elixir, Tailwind, Alpine and LiveView.

These days of DevOps and cloud technologies, we have more of these:

- DevOps Stack[19]
- PLONK[20] Prometheus, Linux/Linkerd, OpenFaaS, NATS, Kubernetes
- SMOKE[21] Serviceful (not just Serverless), Multi-cloud, Open, Kubernetes, Event-driven

And there's more[22] and more[23] stacks. And there's even a comparison platform[24] for stacks.

[17] https://www.danmccreary.com/xrx/
[18] https://changelog.com/posts/petal-the-end-to-end-web-stack
[19] https://devops-stack.io/
[20] https://blog.alexellis.io/getting-started-with-the-plonk-stack-and-serverless/
[21] https://www.triggermesh.com/resources/smoke
[22] https://en.wikipedia.org/wiki/Solution_stack
[23] https://www.webopedia.com/quick_ref/webstack_acronyms.asp
[24] https://stackshare.io/

Migrating from WordPress to Hugo

Adrian Kosmaczewski

2022-04-22

I've been migrating old blog posts (2004-2014) to this blog lately, and you can find them by clicking the "Next" button at the bottom of the index.

I found the corresponding old MySQL database backup at the very bottom of an old backup disk, and to export them into Markdown, I had to do a few things. Maybe these instructions will be useful for somebody else.

Requirements

These are the only technical requirements for this operation:

- Docker or Podman;
- A web browser.

1. Run MariaDB and WordPress

The first thing is to run a combo MariaDB and WordPress; let's use this simple `docker-compose.yml` file:

```
version: '3.3'

services:
   db:
     image: mariadb:latest
     container_name: mariadb
     volumes:
       - db_data:/var/lib/mysql
     restart: always
     environment:
       MYSQL_ROOT_PASSWORD: myrootpass
       MYSQL_DATABASE: wordpress
       MYSQL_USER: wordpress
       MYSQL_PASSWORD: wordpress

   wordpress:
     container_name: wordpress
     depends_on:
```

```
      - db
    image: wordpress:latest
    ports:
      - "80:80"
    restart: always
    environment:
      WORDPRESS_DB_HOST: db:3306
      WORDPRESS_DB_USER: wordpress
      WORDPRESS_DB_PASSWORD: wordpress
      WORDPRESS_DB_NAME: wordpress

volumes:
    db_data:
```

And now:

```
$ docker-compose up
```

Boom, MariaDB and WordPress are running.

2. Import Dump

We can restore the database dump, but before we must do some cleanup:

- First unzip the backup file, which yields a backup file with plenty of SQL commands.
- Then remove those tables from the SQL backup file created by plugins and completely useless in this context, like `wp_twitter`, `wp_woocommerce_*`, `wp_yoast_seo_*`, and others.

But there are a few more issues:

- It turns out that I had exported the database with the wrong collation; hence I edited the backup SQL file to correct encoding errors for old articles, in particular accents in articles in Spanish and French. Le sigh.
- I did not remember the password used to access WordPress back then, so I simply used this tool[1] to create a new hash for a new password of the `admin` user. Then I copied the new hash in the `wp_users` table at the bottom of the dump. Et voilà.
- Finally, to avoid WordPress needlessly redirecting URLs, in the `wp_options` table I changed the entries from the old URL to http://localhost (no `https`! Just plain `http`...) in the `siteurl` and `home` entries.

Before importing the dump, check that you have an empty database called `wordpress`:

```
$ docker exec -it mariadb mysql --user=root --password=myrootpass
Welcome to the MariaDB monitor.  Commands end with ; or \g.
Your MariaDB connection id is 3
```

[1] http://scriptserver.mainframe8.com/wordpress_password_hasher.php

```
Server version: 10.7.3-MariaDB-1:10.7.3+maria~focal mariadb.org binary distribu

Copyright (c) 2000, 2018, Oracle, MariaDB Corporation Ab and others.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

MariaDB [(none)]> show databases;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| mysql              |
| performance_schema |
| sys                |
| wordpress          |
+--------------------+
5 rows in set (0.001 sec)

MariaDB [(none)]> use wordpress
Database changed
MariaDB [wordpress]> show tables;
Empty set (0.001 sec)

MariaDB [wordpress]>
```

And now import the data, just piping from stdin:

```
$ cat data | docker exec -i mariadb mysql --user=root \
        --password=myrootpass --database wordpress
```

This only concerns the text; since I did not find a backup of the `wp-content` folder, many images are missing at the moment. I'll add those as soon as I find them.

3. Access WordPress Console

Now, the moment of truth; open the http://localhost/wp-admin/ URL, and login:

1. Upgrade the database as requested by the latest WordPress; in my case, incredibly enough, it worked without a glitch.
2. Log in as `admin` with the new password, the one hashed in a previous step.
3. Change the theme to "Twenty Twenty" (I personally had no idea what theme I used a decade ago...)

And lo and behold, I could browse my old writing from that point on.

4. Export to Jekyll / Hugo

Finally, I wanted to get all of this content out of the database, in individual files ready for Hugo:

1. I installed the WordPress to Jekyll Exporter[2] plugin (which conveniently enough has absolutely no settings or options)
2. Selected the "Tools" > "Export to Jekyll" option. A zip file downloaded with all posts ready to be imported in Markdown format.

In your `~/Downloads` folder you'll find a nice compressed archive with all of your stuff ready to add to Hugo.

5. Cleanup

Shut down the containers:

```
$ docker-compose down -v
```

The `-v` option removes all volumes attached to the containers.

I'm in the process of reviewing and editing the archives, but many are already in place, stretching back to 2004.

Update, 2022-09-02: I found the backup of the `wp-content` folder, and have restored most images in this website.

Update, 2022-09-16: In case of error, install the WP Debugging[3] plugin and maybe change the value in the "Store uploads in this folder" setting in the "Settings / Media" menu to the following value: `/var/www/html/wp-content/uploads`, so that the Jekyll exporter plugin works properly.

[2] https://github.com/benbalter/wordpress-to-jekyll-exporter/
[3] https://wordpress.org/plugins/wp-debugging/

Open Spaces

Adrian Kosmaczewski

2022-04-29

Somewhere in the 60's, something terrible happened. Someone, out of hatred and ignominy, tore down the walls of his or her office, and showed to the corporate world a new way of working.

The Open Space was born. And since that day, we have had 4 major economic crisis: 1973, 1987, 2000 and 2008.

OK, OK. I am exaggerating. Maybe the concept of open space has nothing to do with the rapidly increasing cycle of economic disasters in the last 40 years. Maybe yes, or maybe not; I am not here to argue. But one thing is sure: our industry changed substantially the day somebody thought that the open space was a good idea for software development teams as well.

Bad Idea

So, here is my personal opinion on open spaces: they are a disaster, a cancer for the industry.

I know that many developers work best in them, but in my personal experience, the best software is made in silence, in peace. No good software can be achieved in the middle of laughter, coffee machines grinding beans, conversations, phone rings, cell phone buzzers, people sneezing or having an impromptu design meeting two desks away.

Yet, this chaotic environment is exactly what the industry offers to those poor souls looking to solve problems. Because our job as software developers is primarily to use code to solve problems; and if one can hardly concentrate for a few hours to solve those problems, one should not expect the solution to be sound, simple or otherwise viable.

Stating the obvious, open spaces are noisy, and that's bad.

But what about teamwork?

The biggest rationale behind this disastrous situation is that open spaces are supposed to increase teamwork. Behind this magic word, managers expect large pools of very expensive souls to solve increasingly complex problems.

The problem is, open spaces are terrible places to solve problems, and since most management gurus talk about open spaces as the solution to all evils in the corporate world, and because open spaces are inherently cheaper than any other office organisation pattern, companies of all sizes end up replicating a model of incoherence and pain that should disappear from the face of Earth.

Possible Solutions

It is naive to believe that these words will shatter the corporate world to its core and remove the open space cancer from the world, as much as I wish this to happen one day. But there are solutions that could be implemented relatively easily that could increase the quality of the software produced in most organisations by an order of magnitude:

1. Keep open spaces only for group activities. Do not allow any developer to perform any kind of "solo" activity in them, but use them in scenarios that involve many members of the team at the same time: team coordination sessions, team building, marketing and design brainstormings, idea gathering, end-user testing sessions, etc.
2. Reserve one or two offices for private work. These offices should be reserved at any time for the development team, so that teams of at most three developers can lock themselves in for a few hours and solve problems in them. When the doors of these offices are closed, nobody should interrupt the developers working in them, and inside of these rooms, silence must be respected: cell phones are set to "airport mode" or at least set to vibrate, and no music is allowed (other than with headphones, that is.)
3. Or, you know, just adopt remote work. This is a trend that the pandemic, and the subsequent "big quit"[1] have accelerated.

These steps are simple, and take into account the fact that software development is **both** a group and an individual activity at the same time. Work environments must provide teams and individuals with the possibility to work as they see fit, and most organisations usually have available offices that could be used this way.

[1] https://en.wikipedia.org/wiki/Great_Resignation

Cardfile.exe

Adrian Kosmaczewski

2022-05-06

I started using Windows 11 recently. It's changed a lot since the last time I used Windows professionally (those were the times of Windows XP, almost 20 years ago). Also, Chocolatey[1] makes it really easy to install software on it.

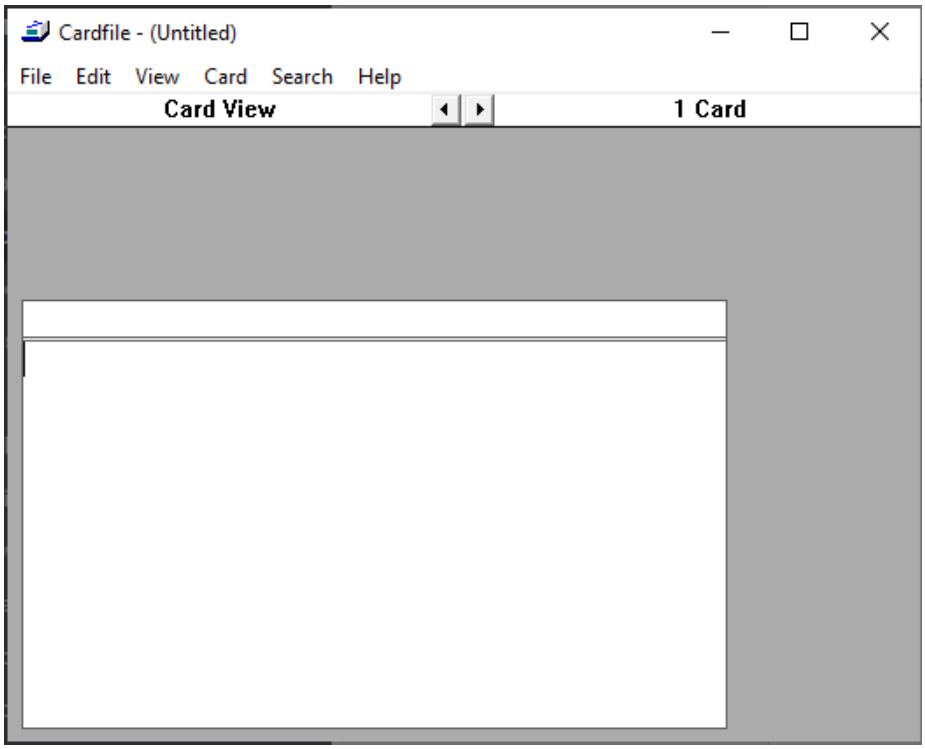
There's one app I miss from the times of Windows 3.1, though; a small app that I loved to use, to collect random thoughts, and to review them quickly and simply. I even used it to prepare exams and stuff like that. I still have those old `*.crd` files laying around, in really old backups somewhere.

That app was `Cardfile.exe`. So I looked for it online (it's not available on Chocolatey) and found it on vetusware[2], and it works perfectly well on Windows 10 and 11: download, unzip, run. I could even change the font to something more modern (in my case, JetBrains Mono[3]) without problem.

But it does not work with Wine[4] on Linux; it crashes when selecting "File / Open" or "Save". Sad, but expected.

It does just one thing, and it does it right. Hit F7 to create a new note, `CTRL + Page Up` and `CTRL + Page Down` to scroll the notes. There used to be a replacement for it called AZZ Cardfile[5], which included `*.crd` file import compatibility, but I haven't tried it.

[1] https://chocolatey.org/
[2] https://vetusware.com/download/Cardfile%20_32-bit_%203.51.1016.1/?id=10104
[3] https://www.jetbrains.com/lp/mono/
[4] https://www.winehq.org/
[5] https://www.azzcardfile.com/


Cardfile - (Untitled)
File Edit View Card Search Help
Card View
1 Card

Swiss Army

Adrian Kosmaczewski

2022-05-13

The war caused by Putin's invasion of Ukraine is shaking the foundations of Swiss neutrality[1]. Needless to mention that neutrality is a main staple of Switzerland, held and cherished since the end of the Napoleonic Wars, just like mountains, milk chocolate, militia, billionaires, overpriced mechanical watches, Victorinox knives, banks, and Nespresso capsules.

The Swiss neutrality is at stake, because the invasion of Ukraine is such an egregious and abject act, not even the Swiss could actually stay put, and rightfully so. And challenging neutrality pisses the hell off our local far-right party, which is always a good thing in my book.

The Swiss government has aligned itself with the sanctions put forward by the European Union, but in spite of that, the USA ain't happy[2]; understandably enough, the Russians ain't happy[3]; NGOs ain't happy[4]; and not even Zelensky is happy[5] with Switzerland, either.

The legendary Swiss Army is unhappy, too, like this very highly ranked Swiss official who was recorded praising Putin on a crowded train[6] and openly criticizing the Swiss government. Oops. Not very neutral, Herr Oberst.

Poor Nestle are also unhappy; first they had to stop selling chocolate in Russia[7] because of consumer pressure, and then they had to scrap their plans to sell cocaine-flavored Nespresso capsules[8]. I guess

[1] https://www.bbc.co.uk/news/world-europe-61320132

[2] https://www.rts.ch/info/suisse/13072407-une-commission-du-congres-americain-accuse-la-suisse-detre-complice-du-dictateur-poutine.html

[3] https://www.rts.ch/info/suisse/13065646-en-colere-contre-la-suisse-la-russie-promet-une-reponse-appropriee.html

[4] https://www.rts.ch/info/suisse/13077531-la-suisse-doit-en-faire-plus-pour-bloquer-les-avoirs-russes-selon-long-public-eye.html

[5] https://www.rts.ch/info/suisse/12952089-volodymyr-zelensky-demande-encore-plus-dengagement-de-la-part-de-la-suisse.html

[6] https://www.msn.com/fr-ch/actualite/other/un-haut-grad%C3%A9-parle-trop-dans-le-train-et-critique-viola-amherd/ar-AAWX5xI?ocid=entnewsntp&cvid=042184a2117647ff8dfba51dba260663

[7] https://www.rts.ch/info/monde/12963947-sous-pression-nestle-reduit-encore-les-produits-vendus-en-russie.html

[8] https://www.bbc.co.uk/news/world-europe-61343037

that's what George Clooney means when he says "what else?"

It is true that Bern has become a real shitshow at the political level. If there's one thing Swiss politicians are bad at, is handling crisis. But I don't blame them for the lack of practice; they haven't had any in 170 years! We have our foreign minister trying to shush those denouncing war crimes[9]. Our head of the military department saying that this war is not such a big risk[10] for our country. The centre parties denounce lack of coordination and coherence[11] in our foreign policy regarding this invasion. What a mess.

The Swiss People's Party[12] ain't happy, either. Because they, like the officer in the train mentioned previously, overwhelmingly support Putin (who, as we all now know, has provided juicy contributions to far-right parties, like in the case of Madame Le Pen in France[13], oh là là!)

Case in point: the billionaire daughter[14] of a major figure[15] of said Swiss People's Party has openly forbidden her employees from talking about the Russian invasion of Ukraine as a "war"[16] and instead to refer to it as a "conflict". Ah, words. Her company, as you might imagine, is busy busy making money money in Russia. It all checks out now. And, to a certain degree, it's understandable: nobody shits in the dining room, innit. As R.E.M. once said[17],

> Shiny happy people holding hands…

There are some Swiss people fighting for Ukraine[18] right now, which happens to be against Swiss law; technically speaking, they could be prosecuted as soon as they return. I wonder if they took their Sturmgewehr 90[19] with them? I'm pretty sure they have their faithful, MacGyver[20]-approved Victorinox Taschenmessern[21] in their pockets, next to their daily portion of chocolate and cookies.

---

[9] https://www.rts.ch/info/suisse/13002882-ignazio-cassis-ce-ne-sont-pas-des-crimes-de-guerre-tant-quun-tribunal-ne-la-pas-decrete.html

[10] https://www.rts.ch/info/suisse/12973081-la-guerre-en-ukraine-ne-represente-quun-faible-risque-pour-la-suisse.html

[11] https://www.rts.ch/info/suisse/13076076-gerhard-pfister-exige-une-politique-de-sanctions-coherente-face-a-la-russie.html

[12] https://en.wikipedia.org/wiki/Swiss_People%27s_Party

[13] https://www.reuters.com/business/media-telecom/false-start-russian-loan-macron-le-pen-debate-takeaways-2022-04-20/

[14] https://en.wikipedia.org/wiki/Magdalena_Martullo-Blocher

[15] https://en.wikipedia.org/wiki/Christoph_Blocher

[16] https://www.watson.ch/fr/suisse/guerre/686609808-martullo-blocher-udc-interdit-le-mot-guerre-a-ses-employes

[17] https://en.wikipedia.org/wiki/Shiny_Happy_People

[18] https://www.rts.ch/info/suisse/13030092-des-dizaines-de-volontaires-venus-de-suisse-au-sein-des-forces-ukrainiennes.html

[19] https://en.wikipedia.org/wiki/SIG_SG_550

[20] https://en.wikipedia.org/wiki/MacGyver_(1985_TV_series)

[21] https://en.wikipedia.org/wiki/Swiss_Army_knife

There was a time when Swiss mercenaries[22] were actually a thing, taking part in major historical events such as the French Revolution of July 1789. I guess the sacrosanct Pontificia Cohors Helvetica[23] is the last remaining officially approved Swiss mercenary army?

The Swiss army has been searching its soul for a long time. I know it, because I've unfortunately been selected to participate in it, a rather somber honor that I would have gladly skipped if it weren't for the fact that the only alternative to enlisting was, well, prison. Yes, the "civil service"[24] was introduced years after it was my turn to wear a uniform. Timing is everything.

As a Swiss citizen, I was drafted in 1992 through a series of physical tests that I failed miserably (seriously, I've never been good at gym), and thus found myself in July 1993 as a member of the rather unremarkable Fliegen und Fliegenabwehr[25] Rekrutenschule 2/243 Dubendorf[26] 93, wearing a Tarnanzug 83[27][28], holding an outdated and overweight Sturmgewehr 57[29], and learning how to do a full Zerlegung and Zusammensetzung[30] in less than 5 minutes without losing any piece in the process. A fundamental and useful skill that I promptly forgot as soon as I could.

I witnessed quite a bit of racism in the Swiss Army. Thanks to extensive Swiss making[31], my unit had people from various different origins (with matching family names): Burundi, Ivory Coast, Chile, Argentina (myself), Poland (myself), Hungary, England, and more. Of course we were called the "foreign legion" (that's cute), and those with darker skin were promptly harassed and insulted (that's not so cute) not only during service, but also in public transportation. Nothing generates warmer patriotic feelings than watching a drunk, two meter high white officer from the special forces of Isone[32] make fun of the dark skin (or the family name) of one of your peers (of yours truly) in front of everyone in the train.

(Record scratch sound clip) Kids: this was the world before smartphones with HD recording capabilities existed.

Besides this blatant discrimination and harassment, being a Swiss

[22] https://en.wikipedia.org/wiki/Swiss_mercenaries

[23] https://en.wikipedia.org/wiki/Swiss_Guard

[24] https://www.vtg.admin.ch/fr/mon-service-militaire/conscrits/service-civil.html

[25] https://en.wikipedia.org/wiki/Swiss_Air_Force

[26] https://en.wikipedia.org/wiki/D%C3%BCbendorf

[27] https://en.wikipedia.org/wiki/TAZ_83

[28] yes, I do have pictures of myself in uniform. I even have one of myself in the cockpit of an F-5 Tiger fighter. No, you won't see them.

[29] https://en.wikipedia.org/wiki/SIG_SG_510

[30] https://www.youtube.com/watch?v=vKa5oifKdzc

[31] https://en.wikipedia.org/wiki/The_Swissmakers

[32] https://en.wikipedia.org/wiki/Special_Forces_Command_(Switzerland)

soldier meant that I had my fair share of Riz Casimir[33], Jass[34] (I prefer playing Truco[35], to be honest), Fassmannschaft, and hours and hours of theory in Swiss German (which I didn't understand).

Such theory tried to teach us how to manipulate the quite useless, clumsy, heavyweight, unusable, and often downright nonfunctional "Funksystem SE-430"[36], which in spite of its name couldn't belong to a universe further away than Jamiroquai[37]'s. I thus became a radio operator, or as German-speaking Switzerland calls them, a "Funker".

So bad was the SE-430, that we often used our shiny new PTT[38] Natel C[39] cellphones to notify our headquarters that our assigned units didn't work (which made us wonder, how did prior generations without cellphones handle this situation? Pigeons? Smoke signals?)

We usually only discovered such shortcoming after having spent 5 hours setting all the equipment up (about 100 kilograms of electronics, antennas, and kilometers of wires covered in cow shit) in some remote part of the country, plugging everything to a portable generator, and trying to send a simple telex message[40] with all of this advanced technology.

The Swiss Army still loves cutting-edge technology. So much in fact, they have decided to beef up their cyberwarfare unit… two months ago[41]. Better late than never, I guess.

We once inadvertently invaded Liechtenstein[42] with our radio system. Really. We planted its antenna system in a field in Kanton Graubünden[43], and we didn't realize there were small stones lost in the grass with the Liechtensteiner flag… anyway, all I'm going to say is that we rarely disassembled a station faster than that day.

But then I heard the true story of a Swiss tank being stopped and sent back home by the French Gendarmerie Nationale[44]… in our case only a few birds witnessed our brief incursion into foreign territory.

(Record scratch sound clip) Kids: this was the world before GPS existed.

Since we're mentioning blunders, let's not forget about the time we

[33] https://www.swissmilk.ch/de/rezepte-kochideen/rezepte/LM200609_53/riz-casimir/

[34] https://en.wikipedia.org/wiki/Jass

[35] https://en.wikipedia.org/wiki/Truco

[36] http://www.armyradio.ch/radio-d/se-430.htm

[37] https://en.wikipedia.org/wiki/Jamiroquai

[38] https://en.wikipedia.org/wiki/Postal_Telegraph_and_Telephone_(Switzerland)

[39] https://en.wikipedia.org/wiki/Natel

[40] https://en.wikipedia.org/wiki/Telex

[41] https://www.swissinfo.ch/eng/politics/swiss-army-to-beef-up-cyber-defence-with-command-centre/47393866

[42] https://en.wikipedia.org/wiki/Liechtenstein

[43] https://en.wikipedia.org/wiki/Grisons

[44] https://en.wikipedia.org/wiki/National_Gendarmerie

switched our emitter to the same frequency as (the now defunct) Swiss Radio International[45], again inadvertently. My apologies to the audience who were greeted with some R2-D2-kind of noise in the middle of their weekly episode of "Yodeling[46] with Reto".

The SE-430 had an "anonymous broadcast" mode, which of course was formally forbidden for everyone to use, but which we played with at two AM in the morning, in the middle of our shift (don't ask). One time we found an old German bible in the barn where we stayed somewhere in Kanton Solothurn, and started broadcasting select chapters of it to all stations, much to the ire and dismay of our superiors. "Am Anfang schuf Gott Himmel und Erde" and stuff like that.

OK, so we had some fun. Not all was negative in the Swiss Army. During trainings we had a mandatory daily box of chocolate available for each soldier, every day. I got free rides all over Switzerland, on top of a very (not) comfortable and very (not) silent Steyr 4x4[47] "Funkwagen". I don't know if you heard this before, but Switzerland is quite a sight. Anyway, I'm digressing.

Oh, and so it happens that I was in that same unit together with Pierre Nebel[48], nowadays a well known political journalist on national French-speaking Swiss television; actually we had also been together to high school in Geneva from 1991 to 93. Very glad to see him doing something he always seemed gifted for: writing and speaking in public.

But I'm digressing again. I remember in July 1993 a colonel came to explain to us how important it was for Switzerland to buy the 34 F/A-18 fighters[49] that the Swiss people had just approved a month earlier[50]. The main reason being the Yugoslav Wars[51] happening at the time. Marketing at its best. Those planes would replace the aged F-5 Tiger[52] fighters of the Air Force.

(While writing this article, I realized that the F-5 was 30 years old in 1993… which is exactly the age of the F/A-18 fighters as this article hits the website. Geez, time flies like a fighter. And soon the F-5s will be replaced by F-35[53] fighters. F-5, F-18, F-35, it all follows a clear progression every 30 years. If the algorithmic progression continues, in 2050 they should buy a couple of F-53.)

I also remember missing the first two weeks of university because of this damn Rekrutenschule. I am actually angrier with my professors, who wouldn't even help me afterwards providing me with copies of whatever they gave to their students during that time, arguing that

[45] https://de.wikipedia.org/wiki/Schweizer_Radio_International
[46] https://en.wikipedia.org/wiki/Yodeling
[47] https://motorfahrer.ch/blog/?p=7806
[48] https://ssrsr.ch/a_la_une/pierre-nebel-jaime-faire-aimer-la-politique/
[49] https://en.wikipedia.org/wiki/McDonnell_Douglas_F/A-18_Hornet#Switzerland
[50] https://www.bk.admin.ch/ch/f/pore/va/19930606/index.html
[51] https://en.wikipedia.org/wiki/Yugoslav_Wars
[52] https://en.wikipedia.org/wiki/Northrop_F-5
[53] https://en.wikipedia.org/wiki/Lockheed_Martin_F-35_Lightning_II#F-35A

“it was my duty” to be there on time. Oh, je suis désolé Monsieur le professeur, I was at my “service militaire” protecting your country. Or something like that. Dammit.

Not only could I not see the end of the tunnel during my Rekrutenschule, I had to endure a few 2- or 3-weeks long Wiederholungskurse[54], once a year from 1994 to 1997, and then again from 2003 to 2006. These sad exercises in futility promptly interfered with my studies, my personal life, my jobs, my everything, every year, once and again, like a neverending curse.

(Record scratch sound clip) A neverending Wiedeholungs curse. Drumroll.

Let’s talk militia, shall we? Need to skip a Wiederholungskurs? Good luck getting your request approved. Those bastards they liked to see you begging for your own fucking time. Oh, but you couldn’t be fired from your job while you’re mobilized, how about that. You can miss exams, though, and that’s your problem, you lazy student.

Hah, “mobilized”; that’s a cute way to describe going for beers dressed in khaki every evening for two weeks year after year. You have no idea how much the Swiss militia supports small pubs in the middle of fucking nowhere. I actually believe that the army owns those bars, because believe me, they are strategically located next to Kasernen, Flugplätze and other military locations far away from any other human settlement, and in many of those you just don’t see anything else but soldiers in and out. Nobody else. Remove the militia system and the GDP of Switzerland would drop by 10 percent.

(Record scratch sound clip) One thing in common of all bars next to military bases is playing Status Quo’s version of In the Army Now[55] every evening. Every. Fucking. Evening.

What else? Oh, did I forget to mention the official shooting exercise to be done every year, too? Ah, yes, here goes another lost Saturday morning per year, to deal with a weapon I was mandated to keep at home, but which I would otherwise not touch, not even with a 10-foot pole.

The whole Rekrutenschule plus Wiederholungskurse thing took too long for my taste. This is the reason why I recommend kids these days to do their whole service in one shot, which takes 9 months, give or take. It’s still a bummer, but at least you won’t need to do any refresher courses afterwards.

My memories of the Swiss Army are those of a clunky, expensive, old, outdated organization, yet one that is so deeply rooted in Swiss culture, it’s going to take a major crisis to change it.

[54] https://www.ch.ch/en/safety-and-justice/military-service-and-civilian-service/military-service/

[55] https://en.wikipedia.org/wiki/In_the_Army_Now_(song)

And maybe, dunno, major crisis we have here and now. Maybe it is time for a redesign of this organization, thanks to those two additional billion Swiss Francs we're going to pay for it[56], and even better, to align it to higher values? The invasion of Ukraine by Russian forces is a good moment to ask ourselves the question: how selfish are we as a country? Or as the American would say: how stupidly do we want to spend our taxpayer money? Supporting bars in the fucking middle of the Berner Oberland playing Status Quo, or actually having a purpose and some integrity?

(Record scratch sound clip) You can guess the answer we'll give as a nation to the question above.

Because, seriously, can we stop this whole militia thing, please? Instead of quantity, let's strive for quality. Having seen what I've seen during my short tenure in such illustrous institution, I'd vote to have a few good men in service rather than a force unfortunately populated with enough drunk[57] rapist[58] nazi nostalgic[59] junkies[60] to make it unpalatable to anyone with a thinking mind.

Instead of such waste, why not having a professional army, made of selected, capable volunteers? My personal take is that it would be a perfect system for Switzerland: knowing the love of Swiss people for their Army, you can be sure its ranks will fill up quickly as soon as the job ads come out. Even better if the pay is good, imagine that. There is a lot of actual patriotism[61] in the Switzerland collective psyche; the militia system is just the wrong way to put it to work.

The Swiss Army must change, Putin must be stopped, and far-right parties must be held accountable and exposed to public scrutiny. All of these events could happen very soon if we want them to happen, as a consequence of the current war in Ukraine.

[56] https://www.rts.ch/info/suisse/13079995-le-budget-de-larmee-suisse-devrait-passer-de-5-a-7-milliards-dici-2030.html

[57] https://www.20min.ch/story/saufen-bis-die-ambulanz-kam-und-wieder-verschwand-573665376805

[58] https://www.blick.ch/schweiz/westschweiz/soldaten-fielen-ueber-soldatin-her-id4880.html

[59] https://www.tagblatt.ch/ostschweiz/rechtsextremismus-heil-hitler-thurgauer-swisscoy-offizier-bleibt-wegen-streit-unter-richtern-bis-heute-straffrei-ld.2214181

[60] https://www.swissinfo.ch/eng/swiss-soldiers-caught-using-drugs/41912140

[61] pay attention to the choice of words here: I say patriotism, not nationalism.

Fortune Apps

Adrian Kosmaczewski

2022-05-20

As part of my work in VSHN[1], I lately prepared a set of demo applications ready to be containerized and deployed in our new product APPUiO Cloud[2].

The applications are available in GitLab[3]. They have all the exact same set of features (not a lot, mind you), while being written in ~~15~~ ~~16~~ ~~18~~ 20 different programming languages.

1. C[4] using the GNU libmicrohttpd[5] system
2. C#[6] using the standard ASP.NET[7] framework
3. Crystal[8] with the Kemal[9] framework (article[10])
4. C++[11] thanks to the Drogon[12] web framework
5. D[13] based on the vibe.d[14] framework
6. Dart[15] using the Conduit[16] framework (article[17])
7. Elixir[18] using the Phoenix Framework[19] (mentioned in a previous article[20])
8. F#[21] using Giraffe[22]

[1] https://www.vshn.ch/
[2] https://appuio.cloud/
[3] https://gitlab.com/vshn/applications
[4] https://gitlab.com/vshn/applications/fortune-c
[5] https://www.gnu.org/software/libmicrohttpd/
[6] https://gitlab.com/vshn/applications/fortune-csharp
[7] https://dotnet.microsoft.com/en-us/apps/aspnet
[8] https://gitlab.com/vshn/applications/fortune-crystal
[9] https://kemalcr.com/
[10] /blog/crystal-is-a-surprise/
[11] https://gitlab.com/vshn/applications/fortune-cpp
[12] https://github.com/drogonframework/drogon
[13] https://gitlab.com/vshn/applications/fortune-d
[14] https://vibed.org/
[15] https://gitlab.com/vshn/applications/fortune-dart
[16] https://www.theconduit.dev/
[17] /blog/dart-is-boring/
[18] https://gitlab.com/vshn/applications/fortune-elixir
[19] https://phoenixframework.org/
[20] /blog/elixir-and-phoenix-framework/
[21] https://gitlab.com/vshn/applications/fortune-fsharp
[22] https://giraffe.wiki/

9. Go[23] based on Gin[24]
10. Java[25] using Red Hat's Quarkus[26]
11. JavaScript[27] based on Express[28]
12. Kotlin[29] based on Ktor[30]
13. Perl[31] with the Mojolicious[32] web framework
14. PHP[33] with the Slim Framework[34]
15. Python[35] using Flask[36] (mentioned in a previous article[37])
16. Ruby[38] with Sinatra[39]
17. Rust[40] using Actix[41] and Askama[42] (metioned in a previous article[43])
18. Scala[44] based on Scalatra[45]
19. Swift[46] using the Vapor[47] framework
20. TypeScript[48] based on Fresh[49] with Deno[50].

Each and every one of these applications are fully containerized, and ready to run. They all offer exactly the same features:

- A simple API returning a `fortune` message, and a random number, on port 8080.
- The API responds JSON when called with `Accept: application/javascript`, plain text when called with `Accept: text/plain`, and just HTML in all other cases.
- It uses server-side rendering, because it's hype again[51], appar-

[23] https://gitlab.com/vshn/applications/fortune-go
[24] https://gin-gonic.com/
[25] https://gitlab.com/vshn/applications/fortune-java
[26] https://quarkus.io/
[27] https://gitlab.com/vshn/applications/fortune-javascript
[28] https://expressjs.com/
[29] https://gitlab.com/vshn/applications/fortune-kotlin
[30] https://ktor.io/
[31] https://gitlab.com/vshn/applications/fortune-perl
[32] https://mojolicious.org/
[33] https://gitlab.com/vshn/applications/fortune-php
[34] https://www.slimframework.com/
[35] https://gitlab.com/vshn/applications/fortune-python
[36] https://flask.palletsprojects.com/en/1.1.x/
[37] /blog/first-web-app-in-rust/
[38] https://gitlab.com/vshn/applications/fortune-ruby
[39] http://sinatrarb.com/
[40] https://gitlab.com/vshn/applications/fortune-rust
[41] https://actix.rs/
[42] https://djc.github.io/askama/
[43] /blog/first-web-app-in-rust/
[44] https://gitlab.com/vshn/applications/fortune-scala
[45] https://scalatra.org/
[46] https://gitlab.com/vshn/applications/fortune-swift
[47] https://vapor.codes/
[48] https://gitlab.com/vshn/applications/fortune-typescript
[49] https://fresh.deno.dev/
[50] https://deno.land/
[51] https://hotwired.dev/

ently.

I could not help myself getting some statistics out of these projects, and making a quick and dirty classification of these programming languages and frameworks. I gathered the following data points:

- The energy ranking provided in this page[52];
- The memory consumption as shown in Docker Desktop;
- The lines of code of the source code of the app, counted by cloc[53];
- Build times and final size of the Docker container as shown in GitLab.

I put all the data in a LibreOffice spreadsheet, multiplied the 5 values above all together into a new `score` column, and ordered everything in ascending order. The build time was interpreted as a timestamp in December 1899 or something like that.

And the results are here; the lower the score, the "better".

Language	Energy Rank	Mem (MB)	LOC	Build (h:m:s)	Size (MiB)	Score
Crystal	3.5	2.29	21	00:02:31	11.83	3.48
Rust	1.03	1.00	66	00:11:20	7.33	3.92
C	1	4.11	102	00:02:30	5.95	4.33
D	3.23	1.87	59	00:02:17	14.33	8.10
Go	3.23	4.77	66	00:02:00	13.22	18.67
Dart	3.83	8.40	76	00:02:35	9.42	41.33
C++	1.34	3.04	79	00:16:06	17.52	62.93
JavaScript	4.45	14.44	58	00:01:44	44.14	198.02
Java	1.98	6.80	133	00:15:09	25.96	488.86
C#	3.14	33.47	88	00:01:48	46.67	539.53
PHP	29.3	24.98	37	00:01:36	26.1	785.34
TypeScript	21.5	14.67	46	00:02:07	43.75	933.03
Python	75.88	19.77	30	00:01:24	22.54	986.22
Ruby	69.91	28.17	20	00:01:22	29.93	1118.83
F#	4.13	31.48	107	00:02:10	55.63	1164.41
Scala	1.98	96.61	70	00:03:08	78.16	2277.27
Swift	2.79	6.15	110	00:15:45	153.86	3173.68
Perl	79.58	38.19	29	00:01:34	105.57	10122.93
Kotlin	1.98	308.80	80	00:04:54	69.57	11579.45
Elixir	42.23	85.03	414	00:03:02	62.21	194810.09

Taking everything into account, I'd consider using **Go, C#, or TypeScript** (in that order) for such an API anytime. With each of these

[52] https://medium.com/codex/what-are-the-greenest-programming-languages-e738774b1957

[53] https://github.com/AlDanial/cloc

three languages, the developer experience is unparalleled, the tooling is excellent, compilation times are super short, the ecosystems are huge and vibrant, and the final results are quite good (by that I mean low energy rank, low memory requirements, low SLOC, and fast build times.)

Why Go, C#, or TypeScript, and not the other programming languages?

- Rust, Java, C++, and Swift build times are too long (any relationship to strong type checking is just a coincidence). But yeah, nobody beats Rust's low memory requirements. To be honest, if Rust's compilation times go down in the future (and I bet they will), it would become the one and only language to consider in this whole list.
- C is… C. Even though the compilation is really quick.
- PHP is… PHP. But the scores of the final product aren't bad at all, don't get me wrong. I kinda like PHP, but wouldn't use it in this context.
- The framework chosen for F# (Giraffe) is a bit behind in terms of documentation. I should have just used ASP.NET here, in which case F# would go up in my personal opinion ranking.
- Elixir, Scala, and Kotlin use too much memory. The interesting thing about Scala and Kotlin is that they are running in a container image with the full Java runtime in it, while the Quarkus container (built with the Java programming language) was just running a self-contained binary, with dramatically lower requirements. No Java runtime, low memory and fast startup; that's precisely Quarkus' selling point.
- Elixir, Python, and Ruby are too high in the energy efficiency index (even though Ruby has the shortest SLOC of all)

Some other observations you might find interesting:

- All container images are based on Alpine[54] except for C and C++ (based on `busybox:glibc`), Java (based on `quarkus-distroless-image:1.0`), Swift (based on Ubuntu 20.04, because Swift does not yet work with musl[55]), and Elixir (based on Debian).
  - The idea was, of course, to have the smallest possible container image with the least possible effort and keeping some readability in the `Dockerfiles`; of course you could make these images smaller, but at the expense of debuggability or understandability. Feel free to use those `Dockerfiles` if you find them useful.
- Python, Typescript, Rust, and PHP use exactly the same HTML templating syntax (and Go's is almost identical).
- Python, Go, Scala, F#, Ruby, and PHP use exactly the same `sprintf()` format, because POSIX and whatnot.

[54] https://www.alpinelinux.org/
[55] https://musl.libc.org/

- Some of these apps were a true PITA to build: in particular, I'll mention Swift. The biggest issue was the low quality and quantity of the Vapor documentation; another issue is the slow compile-test cycles (this was done in Linux, not on a Mac, maybe Xcode can help there, but on Visual Studio Code, no help at all).
  - By the way, the `Dockerfile` of the Swift project is the one generated by the Vapor bootstrap command. Haven't touched it further.
  - Hopefully Swift will be compatible with musl one day (some adventurous people are trying to do it[56] on behalf of Apple), to enable Alpine-based images.
- The easiest ones to build: Python, Ruby, and PHP. Scripting languages are such an easy thing to use, seriously. Although Go, C#, and TypeScript, were also very easy to work with, and with the added benefit of a nice type system on top.
- All images were built for the x86-64 architecture.
- The average SLOC is 96.5 (standard deviation 93.3), and the median 79 (corresponds to the C++ project).
  - The smallest SLOC counts are those of scripting languages: Ruby, Python, PHP, and TypeScript, in that order. They also have very high energy rankings. What you get on one side, you lose on the other.
- The Elixir project contains too much boilerplate code I haven't removed, which contributes heavily to the SLOC. After all, the Phoenix Framework is much more similar in spirit and functionality to Ruby on Rails (or Django) than Sinatra (or Flask).

Finally, I used the code of all of these projects to automatically generate the documentation shown in the Getting Started page[57] of APPUiO Cloud, thanks to the magic of Asciidoctor[58].

Update, 2022-06-17:: I've added a version using Dart[59]. The corresponding data appears in the table above. It gets a very promising 4th place in the table, thanks to a very fast build process, a small final container size, and very low runtime memory requirements. Unfortunately the ecosystem is not very strong around Dart, but otherwise this looks like an excellent choice for a web API.

Update, 2022-07-08: Added new versions in Crystal[60] and Perl[61].

Update, 2022-07-15: Crystal is a real surprise[62]: look at the score in the table above!

[56] https://github.com/MaxDesiatov/swift-alpine

[57] https://docs.appuio.cloud/user/tutorials/getting-started.html#_step_5_deploy_an_application_now

[58] https://asciidoctor.org/

[59] /blog/dart-is-boring/

[60] https://crystal-lang.org/

[61] https://www.perl.org/

[62] /blog/crystal-is-a-surprise/

Update, 2022-08-26: Moved the TypeScript version to use Fresh[63] instead of Node.js, thereby creating a much simpler, JavaScript-only version. And then I added a new version written in the D programming language[64], effectively reaching the number of 20 variations for this project.

[63] https://fresh.deno.dev/
[64] https://dlang.org/

My Biggest Failure

Adrian Kosmaczewski

2022-05-27

Although not my preferred genre, there's a few business books I've read that I reread a few times, and I keep recommending them again and again. "Leading Change"[1] by John Kotter is one of those.

Yet, unfortunately I read it too late. I cannot count with two hands the number of times I made the mistake of not following one or many of the points suggested by this book.

In one particular occasion, however, I remember missing each and every one of the 8 steps for leading change[2]. All of them. At once. One of my biggest business failures, ever.

To make a long story short, I took over a team developing a system which was way over budget and way behind schedule. So I clumsily (and, as it turned out, without any political support from "above") ordered a sweeping change to the complete software development process in the organization, missing all of Kotter's steps, one by one.

1. To begin with, I did not create **a sense of urgency** in the team. Our CEO was worried about the money we were spending and the late time-to-market of our product, but it turns out, they were the only person who was actually worried. Most of the company was actually fine with the current state of affairs. Hence, when my proposal for change came, it was properly dismissed.
2. I had no **guiding coalition** behind me. My CEO was the non-confrontational type, and did not support me in my plan for change.
3. I had skipped the step of actually preparing a **strategic vision** for my initiative, hence those attacking my idea could easily dissolve it in a sea of fear, uncertainty, and doubt.
4. I did not enlist a **volunteer army**; heck, I didn't even get to this point. But even those who could have supported my initiative (namely, the development team) got sidelined by angry middle managers.
5. I could not get to the point of **removing barriers**; I could not even remove the ones that were blocking me from performing.

[1] https://www.kotterinc.com/book/leading-change/

[2] https://www.kotterinc.com/8-step-process-for-leading-change/

6. There were no **short-term** wins to my strategy; which meant that any positive feedback would have taken too long to be taken into account.
7. The whole thing didn't even take off, so let's not even talk about **sustaining any acceleration**. Well, yes, the only acceleration was my departure from the organization.
8. And needless to say, there was no **institution of change**. As I said, I left.

I read Kotter's book one year after the event. It was a revelation, helping me realize all of the mistakes I made in that occasion.

Developers and software engineers tend to belittle and dismiss the work performed by middle and higher managers[3] in organizations. This experience made me realize how complicated, how complex[4], and how difficult it is to drive change in an organization. No wonder so many companies go bankrupt.

I also learnt how thick your skin must be when you're in a position to perform change; how much faith you have to have in yourself (to begin with) and your team. In my case, well; I lacked all of the above. No wonder I failed.

[3] https://deprogrammaticaipsum.com/the-impossible-dialogue/
[4] https://deprogrammaticaipsum.com/complex-vs-complicated/

Live Streaming

Adrian Kosmaczewski

2022-06-03

Lately, I've started to stream some live events on YouTube on behalf of VSHN. I was an absolute live streaming beginner, so I had to learn a few tricks.

Currently, I have a setup that works quite well, running 100% on Linux–Ubuntu 20.04, for the curious amongst you. It should also work without problem on a Mac or Windows, too.

- I'm using OBS Studio[1], of course. This app gathers all inputs from various sources and allows the creation of nice scenes ready to be streamed. At this moment, I'm streaming directly to YouTube. Still, the idea will be in the future to use Restream[2] to broadcast to as many platforms as possible simultaneously: YouTube, of course, but soon also Twitch, LinkedIn, and others.
- In terms of cameras I'm using:
  - A Logitech C920s webcam[3] is mounted on a tripod and connected to the laptop via USB.
  - An iPhone camera (could also be an Android device) using the DroidCam[4] application streams via wifi into OBS thanks to the DroidCam OBS Plugin[5].
  - A DJI Pocket 2[6] camera, live streaming to a containerized RTMP server[7] running on the laptop (connected via wifi), used by OBS Studio as a Media Source. The good thing is that the RTMP stream generated by the DJI Pocket 2 camera also contains audio. The bad news is that when the DJI Pocket 2 is in live streaming mode, its wireless clip microphone doesn't work. Go figure.
- Presentation slides: this is taken directly via an **HDMI splitter** and fed into the laptop using a Sandberg HDMI Capture Link to USB[8], which OBS Studio can use as another video input.

[1] https://obsproject.com/

[2] https://restream.io/

[3] https://www.logitech.com/en-us/products/webcams/c920s-pro-hd-webcam.960-001257.html

[4] https://droidcam.en.softonic.com/

[5] https://www.dev47apps.com/obs/

[6] https://www.dji.com/ch/pocket-2

[7] https://github.com/tiangolo/nginx-rtmp-docker

[8] https://sandberg.world/en-us/product/hdmi-capture-link-to-usb

- Speaker Audio: crucial for an excellent final product, I use a Røde clip microphone[9], with a reasonably long wired extension, directly plugged into the laptop running OBS.

I also usually carry an HDMI extension cable and all the power cords to ensure you don't run out of juice during the transmission. A set of headphones also helps to monitor that the output audio is working properly.

So far, the only issue I've had is that my current HDMI splitter does not work correctly with 2016-20 Macs using the providential USB-C to HDMI dongle. Even the built-in HDMI port in a 2021 MacBook Pro wouldn't work correctly, and lo and behold, we had to use a dongle with that laptop. Like, seriously.

On the other hand, I've tested this setup with Windows and Linux laptops and even with older MacBooks with built-in HDMI ports, and I've had no problem with those. Apple has completely forgotten how to make MacBooks that are good presentation laptops.

If you want to see the result of this setup, here are three videos for you:

- Switzerland GitLab Meetup – 2022-04-27[10]
- Cloud-Native Computing Switzerland Meetup – 2022-05-12[11]
- Liran Tal on NPM Package Security – 2022-05-19[12]

[9] https://rode.com/en/microphones
[10] https://www.youtube.com/watch?v=iYPufKMabYc
[11] https://www.youtube.com/watch?v=TWMW03m614w
[12] https://www.youtube.com/watch?v=rWvBMNmwWEI

On Being a Generalist

Adrian Kosmaczewski

2022-06-10

There is a lot of discussion online these days about the relative benefits (and drawbacks) of being a generalist software developer.

I have always been proud[1] to be a generalist; and I have for a long time[2]. Of course, I have sometimes (somehow) specialized in some technology stack that interested me: .NET from 2000 to 2008, iOS from 2008 to 2019, and Cloud Native things like containers and Kubernetes clusters since 2019. But I always kept an open eye for whatever else was going on in the software world.

This very blog is a reflection of that; back in 2005 and 2006, even though I was working as a .NET developer by day, I was blogging about Ruby on Rails[3], Ubuntu[4], VBScript[5] and C++[6] by night. I can't help it. That's the way I roll. These days I'm interested in Rust[7], TypeScript[8], and even BASIC[9]!

It is precisely this same curiosity what took me to start a magazine like De Programmatica Ipsum[10] with my friend Graham; who, by the way, has also been writing and talking about generalists lately[11]. Being a generalist allows me to see connections and trends across technologies, things that might not be immediately evident to people immersed into a single tech stack every day.

This is the reason why I enjoy so much reading past issues of Byte[12], Dr. Dobb's Journal[13], or keeping up with IEEE Software[14]. Software is

[1] /blog/opinionated/
[2] /blog/on-the-need-of-minimalist-polyglots/
[3] /blog/the-technical-news-of-the-day/
[4] /blog/ubuntu/
[5] /blog/land-of-the-forbidden-maneuver/
[6] /blog/birthdaycard-project/
[7] /blog/more-rust-stuff/
[8] /blog/starting-a-typescript-cli-project-from-scratch/
[9] /blog/basic-standards/
[10] https://deprogrammaticaipsum.com/
[11] https://sicpers.curated.co/issues/9?#start
[12] https://en.wikipedia.org/wiki/Byte_(magazine)
[13] https://en.wikipedia.org/wiki/Dr._Dobb\XeTeXglyph\numexpr\XeTeXcharglyph"0027\relax{}s_Journal
[14] https://en.wikipedia.org/wiki/IEEE_Software

infinitely interesting. After 25 years of practice, I still find it fascinating.

Being a generalist has its perks: it game me the possibility of migrating across galaxies[15], and not only once, but quite a few times in my career. It allowed me to manage multi-language projects like Fortune[16] or Conway[17] from begin to end. It made me appreciate software history; a lot, even: reading how previous generations did our job makes me understand that we've been re-inventing too many wheels, needlessly.

Of course, being a generalist is not always great; in particular, I have to say that it has been indeed very difficult for recruiters to help me find a job in the past. For most of them, I'm not an easy-to-place resource (I know, that phrase sounds horrible). Most of the jobs I have found in my career, it was through my own contacts, or by applying directly to the hiring manager. Recruiters have a hard time with my profile; they definitely prefer specialists. Please understand that I don't blame them, and I can totally understand. I just happen to know the limitations brought by my choices.

Do I recommend other developers to be generalists? Yes, of course I do. I think it's a very rich and enlightening approach, one that will open your eyes to the beautiful world of software and its various facets. It has allowed me to work with people in various fields, with very different backgrounds, and with very different technologies.

What kind of jobs can you do as a generalist software engineer? Well, for starters, it's a great way to become a consultant, particularly in small-to-medium organizations. In such environments you need to be able to take lots of decisions that will have a long lasting impact, and having perspective definitely helps in those cases. Customers need guidance to start new software projects, and being a generalist helps a lot.

You can also be one of the first developers in a young startup, where you need to do a little bit of everything, and you need to coach others to get things done. This is another place where generalists are an asset, in my opinion. Not only will you have to write the code, you will have to choose the programming language, the CI/CD workflows, the methodologies, the testing and deployment harnesses, etc. Being a generalist helps a lot in these cases.

Finally I think that being a generalist can also help you become a good CTO; the best CTOs I worked with, in any case, had perspective and experience in various fields.

But if you do choose to be a generalist, be aware that your market value might erode a bit ; not everywhere, but in some select places you might be interested to work in. If that's the case, you'd better

[15] /blog/the-developer-guide-to-migrate-across-galaxies/

[16] /blog/fortune-apps/

[17] /blog/polyglot-conway/

follow the instructions of the excellent “Positioning Manual for Indie Consultants”[18] by Philip Morgan, become a super specialist in some tech stack, and enjoy the benefits that approach could bring.

In short: there is no “better” approach, but simply ones that work better for you, in your context, and following your own taste. You will have to find a way, and I suggest to suppress all those voices around you, telling you to specialize or perish. Develop your own taste, find what you like about software, and follow it.

PS: you might have noticed that I have not talked about the famous “T-shaped skills” approach[19]; it’s up to you to decide whether or not it’s worth a road to follow. I, for one, am not so sure about it, and still prefer to grow my overall curiosity.

[18] https://philipmorganconsulting.com/positioning-manual/
[19] https://en.wikipedia.org/wiki/T-shaped_skills

Dart is Boring

Adrian Kosmaczewski

2022-06-16

Lately, I've been playing with Dart[1], the programming language powering the cross-platform Flutter[2] UI framework. I've added a Dart implementation[3] to my collection of (now 21) programming languages in the Conway project[4], and another[5] to the collection[6] of sample Fortune web APIs.

As a language, Dart certainly fulfills the promise of the "no surprises" mantra. I could go as far as to say that Dart is essentially boring. Don't get me wrong; I think that's a good thing. The language does what is expected to do, without frills or surprises, but sadly without much fun, either.

Conway

If you look at the program's code[7], there's hardly anything you can't understand immediately.

```
import 'dart:io';
import 'coord.dart';
import 'world.dart';

void clrscr() {
  print("\x1B[2J\x1B[0;0H");
}

void main() {
  var alive_cells = World.blinker(new Coord(0, 1));
  alive_cells.addAll(World.beacon(new Coord(10, 10)));
  alive_cells.addAll(World.glider(new Coord(4, 5)));
  alive_cells.addAll(World.block(new Coord(1, 10)));
  alive_cells.addAll(World.block(new Coord(18, 3)));
```

[1] https://dart.dev/
[2] https://flutter.dev/
[3] https://gitlab.com/akosma/Conway/-/tree/master/Dart
[4] /blog/polyglot-conway/
[5] https://gitlab.com/vshn/applications/fortune-dart
[6] /blog/fortune-apps/
[7] https://gitlab.com/akosma/Conway/-/blob/master/Dart/src/conway.dart

```
  alive_cells.addAll(World.tub(new Coord(6, 1)));
  var world = new World(30, alive_cells);
  var generation = 0;
  clrscr();
  while (true) {
    generation += 1;
    print("$world");
    print("Generation $generation");
    world = world.evolve();
    sleep(Duration(milliseconds: 500));
    clrscr();
  }
}
```

Dart seems to be exclusively designed for one and only one thing: to power the Flutter[8] framework. Flutter allows developers to build applications for iOS, Android, Linux, Windows, Mac, and the web with just one codebase, and I know a few passionate developers who are excited to use it.

As expected of any Turing-complete language, people use Dart for many other things[9]. For example, there are a few web frameworks like Conduit[10] and Jaguar[11]. I will talk about Conduit later in this article.

I suppose Flutter is much more interesting than Dart, to be honest. This experiment had more to do with understanding Dart as a language, to get a quick overview of its capabilities and syntax, and nothing else. In terms of verbosity, for example, Dart is more or less in the same league as Java and PHP.

The Dart developer experience was good; I used the Dart Visual Studio Code extension[12], which provided excellent feedback during coding.

Dart also includes lots of “modern”[13] programming language features:

- It’s open-source[14];
- Generics;
- Ruby-style mixins;
- Garbage collection;
- `async` and `await`;
- Functional constructs;
- Null safety[15];
- It has integrated unit testing;

[8] https://flutter.dev/
[9] https://github.com/yissachar/awesome-dart
[10] https://www.theconduit.dev/
[11] https://github.com/Jaguar-dart/jaguar
[12] https://marketplace.visualstudio.com/items?itemName=Dart-Code.dart-code
[13] https://deprogrammaticaipsum.com/the-great-rewriting-in-rust/
[14] https://github.com/dart-lang
[15] https://dart.dev/null-safety

- There's an online playground[16] to test the language;
- It has type inference;
- Features a "batteries included" library of data structures and algorithms;
- It provides an automatic source code formatting tool;
- And it compiles code to native binaries.

All very modern indeed, as expected by a language designed by Lars Bak[17].

On the other hand, it does have class inheritance, variables are not immutable by default, and it requires semicolons. Boohoo. I'm kidding of course. Sense the irony.

Fortune

I also implemented the Fortune application using Dart[18], and I have a few observations:

- All those web API frameworks I found to create such a project lack traction and community involvement; low number of releases, not a lot of open issues, not a lot of PRs, etc. Some of them are simply discontinued. I guess Dart is not the most popular language to use in this context at the moment.
- I used Conduit[19], itself a fork of Aqueduct[20] which had been archived[21] by its owners.
- Conduit does not include a template library, so I just used the mustache_template[22] package instead, and it worked perfectly well.

The resulting router[23] is quite straightforward:

```
@override
Controller get entryPoint {
  final router = Router();
  final source = File("templates/fortune.html").readAsStringSync();
  final template = Template(source);
  final rng = Random();
  final hostname = Platform.localHostname;
  const version = "1.0-dart";

  router.route("/").linkFunction((request) async {
    var message = "";
    await Process.run('fortune', []).then((ProcessResult pr) {
```

[16] https://dart.dev/tools/dartpad
[17] https://en.wikipedia.org/wiki/Lars_Bak_(computer_programmer)
[18] https://gitlab.com/vshn/applications/fortune-dart
[19] https://www.theconduit.dev/
[20] https://aqueduct.io/
[21] https://github.com/stablekernel/aqueduct
[22] https://pub.dev/packages/mustache_template
[23] https://gitlab.com/vshn/applications/fortune-dart/-/blob/master/lib/channel.dart

```
      message = pr.stdout.toString();
    });
    final random = rng.nextInt(1000);
    final json = {
      'number': random,
      'message': message,
      'version': version,
      'hostname': hostname
    };
    final acceptHeader = request.acceptableContentTypes.first;
    if (acceptHeader.toString() == "text/plain") {
      final text = "Fortune $version cookie of the day #$random:\n\n$message";
      return Response.ok(text)..contentType = ContentType.text;
    } else if (acceptHeader.toString() == "application/json") {
      return Response.ok(json)..contentType = ContentType.json;
    }
    final html = template.renderString(json);
    return Response.ok(html)..contentType = ContentType.html;
  });

  return router;
}
```

Building the Docker container was a bit more delicate, as always; the thing is that even though Dart can compile to native binaries using the `dart compile exe bin/main.dart -o bin/main` command, in the case of Conduit this doesn't work; one must use the `conduit` CLI instead, as explained in this issue[24] by the author.

We then use that binary directly in an Alpine-based image, with the required `fortune` package, and the final product is really small and fast to boot.

All things considered, the final score of the Dart-based container image is impressive: very small final container image size; very low memory consumption; and very fast build times. Check the results in the table on this page[25] for details, where Dart appears now in the 4th place, behind Rust, C, and Go, and before C++.

Unfortunately, the fragmented and fragile perception I got of the Dart ecosystem would make me wary of choosing it for a production project, and for this reason, I won't change my final recommendation:

> Taking everything into account, I'd consider using **Go, C#, or TypeScript** (in that order) for such an API anytime.

[24] https://github.com/conduit-dart/conduit/issues/90#issuecomment-1126872018
[25] /blog/fortune-apps/

Conclusion

All in all, Dart is boring, and that's its greatest feature. It does what it says it will do. It has an excellent library and the support of a major vendor like Google behind. I would love to see more activity on other projects than Flutter around Dart, I think the language has lots of potential, and I can now understand why some people are so bullish about it. But the ecosystem, compared to those of other languages like Go, C#, or TypeScript, is still quite weak.

The Recruitment Lottery

Adrian Kosmaczewski

2022-06-24

Finding a job as a software engineer is a lottery. You will find lots of contradicting advice online, and what I'm doing here is adding my grain of salt to this big cacophony that is the world of technical recruiting.

Younger engineers sometimes contact me on Twitter and ask me for advice about how to find a (first, second, nth) job. Let's analyze some common factors considered crucial to recruiting, one by one.

Open Source Portfolio

Some people say your open-source software contributions are a key factor to get a job in the software industry; your pull requests and number of followers on GitHub will get you the job, not your resume[1].

Bullshit, I say, and Marcin agrees[2].

There is no single recipe to landing a job; there is no single factor that will consistently make a difference in your job search. Open Source is not one of them.

Let's make this clear:

Every. Single. Company. And. Their. Recruitment. Processes. Are. Different. And. Equally. Flawed.

Write that down and memorize it. Done? Good, because it's a lottery. Every single interview process will be different.

Back to the subject of code, some companies might care about your FOSS code contributions; but they won't probably do so for the reason you think. They might be more interested in your capacity to work for free; if you contributed so many pull requests for nothing, imagine all the code you're going to write for us for this meager salary. That's what they call passion at work™ ®©.

Other companies might look at the actual code you've written, cloning the repos and seeing your coding style, the number of tests you've written (or not), the documentation you might have written (or not),

[1] https://twitter.com/mfts0/status/1529475379949457408

[2] https://twitter.com/krzyzanowskim/status/1529765652910772224

stuff like that. I did that when hiring a while back, and it is an interesting indicator, but beware, lots of people game recruiters with useless code they've copied from somewhere else, just to get the foot in the door.

Dear reader: don't be that candidate.

So I would review the FOSS code as a criterion, but would not consider it the definitive one. In particular, it does not work with engineers who have worked in corporations not sharing code (they are the majority, still), like banks, finance, healthcare, or other industries where trade secrets are never disclosed. Not all companies are into open-sourcing their stuff. What? Shocker!

Don't dismiss an otherwise excellent engineer because they don't have projects on GitHub. And gosh, don't expect or pretend that they do that in their free time, either. People have a life to live, and that goes beyond writing code, geez.

Anyway, it's a lottery.

Resumes

Some companies don't give a shit about your code and will be exclusively drilling your resume during interviews. This is another category with two very distinct subcategories.

The first subcategory is the one that looks at what you've done and judges you for that. That's usually fine, I guess.

The second subcategory judges you for your gaps, that is, for what you haven't done. There's a big swath in the recruiting population that simply does not want to see gaps on resumes.

I'm serious, some get angry. Imagine a person you don't know and you've never seen (and hopefully, you'll never meet again, like, ever) starts belittling you because you took a year to travel around Patagonia 20 years ago.

I have a few gaps on my resume, so these people have always been easy to spot. I don't want to work with such people. So all in all, yeah, I'd recommend you take a sabbatical, just for the sake of having a gap or two in your resume. Believe me, it helps to spot assholes. And it's awesome, you get to travel the world and see stuff and meet people. Sabbaticals are a win-win-win.

Regarding the cover letter; some companies just wouldn't even consider your application without one, some others just don't even read them, and I knew a few that downright dismissed candidates because they wrote one.

So, again, it's a lottery.

Tech Questionnaires and Assignments

Some companies are going to drill you down with technical questions. This is, I admit, the point where I wrap up the interview and leave. I mean, I leave the room at that point.

Judging a person for the amount of stuff they can memorize is ridiculous. Particularly when they can find that information immediately with a DuckDuckGo or Google search.

What is the point of being happy about a candidate who could reply to your questions correctly? What if you had the wrong answer and the candidate gave you the correct one? (Been there, experienced that, and not only once.) Are you going to ditch them because of your ignorance? Really?

Particularly in Switzerland, the exercise in futility of technical interviews makes no sense. You know what? Because there's a trial period of three months at the beginning of every employment contract. You have three months to get rid of a person in just a week if it doesn't work. So what is then the point of going through an exhausting process of weeks and weeks and judgments and ghostings that makes everybody unhappy and that has a higher noise-to-signal ratio than expected?

Memorizing stuff means nothing. Reversing the proverbial linked list on the proverbial whiteboard means nothing.

People can (should) offer much more to an organization than just raw knowledge. Like teamwork; empathy; respect; proactivity; insight; experience; context; and so on. Don't get me wrong; I do understand the need to gauge people's knowledge, to a certain extent. But questionnaires are simply the wrong approach.

For that reason, if all else fails, I would recommend doing a **paid exercise**. If you, as a business, are so keen to find the best of the best, then hire them as freelancers for 2 or 3 hours, give them an exercise, and then pay them for that (yes, pay them, a decent hourly rate). Instead of a series of stupid questions whose answers can be found in books, just ask them to do something. Not on their time, and not for free; invite them to your office, or work with them on a Zoom call, and see them at work. Even better, have one of their future colleagues with them, and see how they react and interact. Do they ask questions? Do they propose "out-of-the-box solutions"? If they do, maybe you've found a gem.

And even if you didn't, pay them for the time, say thank you, and please, please, **please: whatever the result of this process, don't ghost them. Just don't.**

But since I doubt recruiters are reading this post, this is my message to you, dear candidate: it's a lottery. Don't do coding exercises for

free; if a company gives you a “4 hours assignment”[3] to complete, tell them your hourly rate and send them an invoice at the end.

As Ray Liotta[4] and later Mike Monteiro[5] both said, “Fuck you. Pay me.”

Yes, I think that candidates should be compensated for their time spent fulfilling exercises which, in many cases, end up being used by the very businesses they applied for. Unethical businesses? What? Shocker!

I tell ya, it’s a lottery.

Process

Some companies are going to tell you upfront everything about the hiring process; the length, the number of interviews, the number of people to talk to, and also the budget for the position. This is how it should be.

But even if all seems (or seemed) to be going well during the recruitment process, it doesn’t mean anything!

Right after the 3rd interview of a series of 7, you could get an e-mail that says that the position has been filled internally, or that you’re not a good “culture fit” (whatever that means), or that the position has just been canceled altogether, or that the company moved into a hiring freeze (which is, by the way, happening a lot these days[6].) This could even happen after you passed all the interviews and received an offer!

In my case, I even had the experience of arriving on my first day of work and realize that the job I had applied for did not exist anymore in the company. I wrote about that experience[7] 12 years ago.

Oh, and if you hear the words “we are family” run away for dear life. You can find more information about that in this blog[8]. There is always a hidden agenda[9] in those words. Ask yourself this question: who do you want to work with?[10] Remember that programming is a clerk’s job[11].

If you decide to go kick off the interview process, here are a few interview tips[12] for you from yours truly.

But again, I insist; it’s a lottery.

---

[3] https://news.ycombinator.com/item?id=31812864
[4] https://www.youtube.com/watch?v=3XGAmPRxV48
[5] https://www.youtube.com/watch?v=jVkLVRt6c1U
[6] https://twitter.com/GergelyOrosz/status/1523004583160819712
[7] /blog/welcome-to-the-company/
[8] /blog/we-are-family/
[9] /blog/you-founder-and-your-secret-agenda/
[10] /blog/who-do-you-want-to-work-with/
[11] /blog/the-truth-be-told/
[12] /blog/job-interviews/

Money

If you are not told the salary of the position or the complete details of the upcoming interview process in the first interaction with your prospective employer (or its deputy recruiter), just walk away. It does not make any sense to embark on a journey of 5 or 6 or 4 or 10 interviews (yes, you read right) if you don't know how much money we're talking about at the end.

On the other hand, if they have the courage of asking you if you know how much you will cost them as an employee, again, run away for dear life redux. Read more about this particular case on this blog[13].

And if you ask and they are offended or surprised because of your materialistic views, walk away. Keep in mind that a job, it's a job, that is, a job; in other words, it's a business relationship. **You give your brain CPU time in exchange for cash.** That is all. There's no "passion" in there, there are no "ninjas" in there. The whole idea of you working somewhere is to be able to make cash. If we didn't need to make cash to live, very few people would work, or at least not according to the rules of the current broken state of modern capitalism.

So, whatever company you'd like to interview for, please make sure to get the following 2 pieces of information at the first interview (actually, you shouldn't ask for them, and they should tell you; and if they don't, write that down and remember it):

1. Length and description of all steps in the recruiting process.
2. Amount of yearly salary, and eventual bonuses, for the position.

Remember that it's a lottery and that some companies might not provide this information; to the first question, because it's confidential (why would that be the case, that's beyond me), and to the second, because they argue it demonstrates "a lack of passion in you", dear candidate aware of the realities of our modern economy.

Lottery

Once again, répétez avec moi: the recruiting process is a lottery. Every company is different, yet they all believe they got the best process. All of them suck equally, none has it completely right, and given this reality, I'd pick a company that is upfront and humble and even a bit messy anytime.

An upfront and humble company will be respectful and clear with you and your candidacy from day one. They will get back to you when they say they will. They will even provide some useful feedback if the answer is negative. They will value your time and patience, and retribute it to you if they ask for a practical exercise. They will not reply to you with dry copy-pasted templated texts.

[13] /blog/the-wrong-question/

I pondered in this blog the need to rethink the corporate world[14] 15 years ago (!), yet here we are, still.

So, remember: as in any lottery, you have more chances of losing than winning.

PS: if you would like to read some personal interview anecdotes, you will find them in this De Programmatica Ipsum article[15] written in April 2019.

[14] /blog/rethinking-the-corporate-world/
[15] https://deprogrammaticaipsum.com/tales-of-the-interview/

Various Flatpak Tips and Tricks

Adrian Kosmaczewski

2022-07-01

I use Flatpak[1] a lot, on my work and personal laptops. I had to find a few tips and tricks to make it work the way I wanted, using all the apps I prefer.

This page summarizes the things I've done to make Flatpak work for me, and hopefully, they'll be useful to you too.

Toggle External Editing in Joplin

Due to sandboxing, when installing Joplin[2] via Flatpak, the "Toggle External Editing" command does not work by default. There was an issue[3] about this, and now the project README[4] explains how to make it work.

If you want to use an app not installed through Flatpak (for example, in this case, Visual Studio Code), you have to open the Joplin settings and use the following parameters in the "Text editor command" section of the General tab:

- Path: `/bin/flatpak-spawn`
- Arguments: `--host /usr/bin/code`

However, if you would like to use a Flatpak-installed editor such as Apostrophe[5] or Typora[6], use the following parameters:

- Path: `/bin/flatpak-spawn`
- Arguments: `--host /bin/flatpak run --filesystem=xdg-config/joplin-desktop io.typora.Typora`

Of course, the command above requires you to install the corresponding app:

```
$ flatpak install flathub io.typora.Typora
```

[1] https://www.flatpak.org/
[2] https://joplinapp.org/
[3] https://github.com/flathub/net.cozic.joplin_desktop/issues/9
[4] https://github.com/flathub/net.cozic.joplin_desktop
[5] https://gitlab.gnome.org/World/apostrophe/
[6] https://typora.io/

If you prefer Apostrophe, it's the same but with `org.gnome.gitlab.somas.Apostrophe` as app identifier.

Calibre Night Mode

If you install Calibre[7] with Flatpak, unfortunately, it does not read the current system theme at startup (or at runtime) and stays stubbornly white at all times of the day, even if the other apps are all dark and cozy.

If you would like to use it in night or dark mode, type the following command:

```
$ flatpak override --user --env=CALIBRE_USE_SYSTEM_THEME=0 \
    --env=CALIBRE_USE_DARK_PALETTE=1 com.calibre_ebook.calibre
```

After this, launching Calibre will always use the dark mode. Run the same command with `CALIBRE_USE_DARK_PALETTE=0` or `CALIBRE_USE_SYSTEM_THEME=1` if you would like to go back to the default. Unfortunately, the latter always defaults to the non-dark version, even if the system has switched to dark more.

Update, 2022-07-22: Calibre has been recently updated to 6.0 and it does not take into account any more the environment variables to select day or night mode; you can now select the mode directly in the preferences.

Signal Desktop Start in Tray

Signal Desktop[8] for Linux has a toggle to start minimized in the system tray, but unfortunately, it's not exposed in the preferences (unlike its Windows counterpart). The trick consists in creating a shell shortcut and adding a parameter in the command. This should work for both GNOME[9] and KDE[10].

First, create a duplicate of the shell shortcut:

```
$ cp /var/lib/flatpak/exports/share/applications/org.signal.Signal.desktop ~/.l
$ vim ~/.local/share/applications/org.signal.Signal.desktop
```

Then add the `--start-in-tray` option in the Exec line:

```
[Desktop Entry]
Name=Signal
Exec=/usr/bin/flatpak run --branch=stable --arch=x86_64 --command=signal-deskto
Terminal=false
Type=Application
Icon=org.signal.Signal
```

[7] https://calibre-ebook.com/
[8] https://signal.org/download/
[9] https://www.gnome.org/
[10] https://kde.org/

```
StartupWMClass=Signal
Comment=Private messaging from your desktop
MimeType=x-scheme-handler/sgnl;x-scheme-handler/signalcaptcha;
Categories=Network;InstantMessaging;Chat;
X-Desktop-File-Install-Version=0.26
X-Flatpak-RenamedFrom=signal-desktop.desktop;
X-Flatpak=org.signal.Signal
```

The desktop environment picks up the new command automatically, and it's ready to use. This is explained here[11].

The Flatpak Filesystem

In all of the explorations above, knowing how Flatpak organizes files is very useful. For example, that's how you find GIMP's configuration files, located in `~/.var/app/org.gimp.GIMP/config/GIMP/2.10`; which can be useful, for example, if you want to change the keyboard shortcuts[12] to be more similar to Photoshop's.

There's a wiki page about it[13] in the GitHub project, and it looks a bit like this:

```
.local/share/flatpak/
├── repo
│   ├── config
│   ├── objects
│   │   ├── 00
│   │   ┊
│   │   └── ff
│   ├── refs
│   │   ├── heads
│   │   └── remotes
│   │       └── test-repo
│   └── state
├── runtime
│   └── org.gnome.Platform
│       └── x86_64
│           └── 3.14
│               ├── active -> 71885e962e0daa8635cc7f33…
│               ├── 71885e962e0daa8635cc7f33…
│               │   ├── files
│               │   │   └…
│               │   └── metadata
│               └── origin
├── app
│   └── org.gnome.GEdit
│       ├── current -> x86_64/master
```

[11] https://github.com/flatpak/flatpak/issues/2913#issuecomment-915300401

[12] https://www.linuxuprising.com/2018/11/configure-gimp-210-to-use-photoshop.html

[13] https://github.com/flatpak/flatpak/wiki/Filesystem

```
│       └── x86_64
│           └── master
│               ├── active -> cf22ec5c375bba629b33385a…
│               ├── cf22ec5c375bba629b33385a…
│               │   ├── export
│               │   │   └…
│               │   ├── files
│               │   │   └…
│               │   └── metadata
│               └── origin
└── exports
    └── share
        ├── applications
        ├── dbus-1
        └── icons
.var/
└── app
    └── org.gnome.GEdit
        ├── data
        ├── config
        └── cache
```

Update, 2022-07-22: I also discovered `flatpak uninstall --unused` to remove unused runtimes.

The New Microsoft

Adrian Kosmaczewski

2022-07-08

Microsoft is a big, big, big name in our industry. No matter what they do, everybody notices. Whether it's good or bad, useful or ridiculous, big or small, it never goes by unnoticed.

I have written a lot about Microsoft, either in this blog[1] or in De Programmatica Ipsum. In the latter we literally had an issue dedicated to it[2]. In other editions, I chose to write about their history of EULAs and patents[3] and into their newfound love for JavaScript[4].

Most importantly, I pondered about its motivations nowadays[5].

In any case, it's impossible to ignore them. Particularly in my case; for a large chunk of my career, I actually made a living writing code in whichever programming language they bundled with Visual Studio.

And then, one day in February 2014, a previously unheard person (for me) named Satya Nadella[6] became the third CEO in the company's history.

And in the space of a few years after that, Microsoft joined the Linux Foundation. They bought Xamarin, LinkedIn, and GitHub. Azure grew to become a serious competitor (maybe the only one) to Amazon AWS. Microsoft Teams[7] becomes the most hated product during the pandemic. Microsoft Edge ditches its engine for Chromium. Their Twitch channel[8] grows to 12'000 followers, effectively deprecating Channel 9[9]. Surface laptops and tablets started appearing in Grey's Anatomy[10] in lieu of iPads and MacBooks.

Understandably, enough, its stock price followed.

---

[1] /tags/microsoft/

[2] https://deprogrammaticaipsum.com/issue-37-microsoft/

[3] https://deprogrammaticaipsum.com/the-conquest-of-code/

[4] https://deprogrammaticaipsum.com/innovationscript/

[5] https://deprogrammaticaipsum.com/where-does-microsoft-want-to-go-today/

[6] https://en.wikipedia.org/wiki/Satya_Nadella

[7] https://en.wikipedia.org/wiki/Microsoft_Teams

[8] https://www.twitch.tv/microsoftdeveloper

[9] https://en.wikipedia.org/wiki/Channel_9_(Microsoft)

[10] https://en.wikipedia.org/wiki/Grey\XeTeXglyph\numexpr\XeTeXcharglyph"0027\relax{}s_Anatomy

For developers, many names mark this new era; let's enumerate some of them. TypeScript[11] to bring some order on the frontend; F#[12] to make functional developers happy; .NET[13] celebrating its 20 years while finally available on Linux and Mac; .NET MAUI[14] for all your user interfaces; Visual Studio Code[15] to write all code; SQL Server for Linux[16] because why not; GitHub[17] to own all code; npm[18] to own all JavaScript; Scott Hanselman[19] to teach us everything we need to know.

To say that the company has changed in the past 8 years is an understatement.

[11] https://www.typescriptlang.org/
[12] https://fsharp.org/
[13] https://dotnet.microsoft.com/en-us/
[14] https://docs.microsoft.com/en-us/dotnet/maui/what-is-maui
[15] https://code.visualstudio.com/
[16] https://docs.microsoft.com/en-us/sql/linux/sql-server-linux-overview?view=sql-server-ver16
[17] https://github.com/
[18] https://www.npmjs.com/
[19] https://www.hanselman.com/

Crystal is a Surprise

Adrian Kosmaczewski

2022-07-15

I blogged about Dart a few weeks ago, and I said it was refreshingly boring[1]. I am probably late to the party (well, evidently I am, as it's been around since 2014), but I discovered Crystal[2] recently, and it is not only boring but also surprising in many delicious ways.

(And yes, I like boring technology. A lot.)

If I say that Crystal is "compiled Ruby" I'm oversimplifying things. Well, ahem, am I? At last year's Crystal Conference[3], celebrating the release of Crystal 1.0, the keynote speaker was none other than Yukihiro 'Matz' Matsumoto[4] himself. (If you don't know who he is, he invented Ruby.)

Beyond the syntactic similarities, Crystal is quite an improvement over Ruby. It includes many modern features[5], such as: null safety, a package dependency system, a build tool, type inference, garbage collection, a robust type system, a built-in preprocessing macro system, generics, Go-like fibers and channels for multiprocessing, and a lightning-fast compiler (based on the LLVM toolchain) that outputs lightning-fast code. All very modern.

Another nice touch is that the compiler can generate statically linked binaries not needing anything else to run; this feature makes Crystal a great candidate for writing containerized ("Dockerized") applications.

And finally, the thing that surprised me the most: it comes from my childhood neighborhood. The company behind it[6] has its headquarters merely 10 blocks away from the location where I grew up as a kid[7], in Vicente López, north of the city of Buenos Aires. Call me nostalgic, but it struck a chord.

[1] /blog/dart-is-boring
[2] https://crystal-lang.org/
[3] https://crystal-lang.org/conference/
[4] https://github.com/matz
[5] https://deprogrammaticaipsum.com/the-great-rewriting-in-rust/
[6] https://manas.tech/
[7] /blog/thirty-years/

Fortune

As usual, I created yet another example of the Fortune application[8], available for your exploration in GitLab[9]. I used the Kemal[10] framework, and to be honest, I quite literally copied the source code of the Ruby version[11], changed a few things, and voilà, done.

From a syntactic point of view, Crystal is uncannily similar to Ruby. Let's compare both "Fortune" apps written with them, starting with Crystal.

```
require "kemal"

version = "1.2-crystal"
hostname = `hostname`

get "/" do |env|
  message = `fortune`
  number = rand(1000)
  accept = env.request.headers["Accept"]
  if accept == "application/json"
    obj = { :version => version,
            :hostname => hostname,
            :message => message,
            :number => number }
    obj.to_json
  elsif accept == "text/plain"
    "Fortune %s cookie of the day #%d:\n\n%s" % [version, number, message]
  else
    render "src/views/fortune.ecr"
  end
end

Kemal.config.port = 8080
Kemal.run
```

Look at the Ruby version for comparison:

```
require "sinatra"
require "sinatra/reloader" if development?

set :port, 8080

get '/' do
  @version = "1.2-ruby"
  @hostname = `hostname`
  @message = `fortune`
  @number = rand(1000)
```

[8] /blog/fortune-apps/
[9] https://gitlab.com/vshn/applications/fortune-crystal
[10] https://kemalcr.com/
[11] https://gitlab.com/vshn/applications/fortune-ruby

```
  if request.accept? 'text/html'
    erb :fortune
  elsif request.accept? 'application/json'
    obj = { :version => @version,
            :hostname => @hostname,
            :message => @message,
            :number => @number }
    JSON.generate(obj)
  elsif request.accept? 'text/plain'
    "Fortune %s cookie of the day #%d:\n\n%s" % [@version, @number, @message]
  end
end
```

Seriously... what!?

But the biggest surprise, aside from the syntax, was the result of the web framework comparison I presented in this blog post[12]. Based on energy ranking, memory required at runtime, number of lines of code, build time, and size of the final container image... **the big winner in the ranking is Crystal!** I'm planning an update to this ranking including other criteria, such as ecosystem size, average latencies, and other important points. But this clearly was an unexpected result, and it caught me completely off-guard.

Crystal compiles quickly into small and fast binaries. What's not to love?

Conway

I also created the Crystal version[13] of the Conway app[14], and again, I quite literally copied the code of the Ruby version, and fixed the few following things until the app worked:

- Use `require` instead of `require_relative`;
- Implement the `hash(hasher)` method in the `Coord` class, to be able to use it as a `Hash` key;
- Added type information in some method signatures and when instantiating generic collections;
- Replace `attr_reader` with `getter`;
- Implement `to_s(io : IO)` for performance;
- Change the handling of `SIGINT` signals in the main app;
- Replace `for in` range loops with `(0...bound).each`;
- Override `==()` instead of `eq?()` for equality.

And I promise, that was it. It must have taken me no more than 40 minutes until it worked. Many of the changes enumerated above have to do with the fact that Crystal precluded some dynamic aspects of the Ruby runtime for performance reasons. A very wise choice.

[12] /blog/fortune-apps/
[13] https://gitlab.com/akosma/Conway/-/tree/master/Crystal
[14] /blog/polyglot-conway/

The only caveat I could detect in the developer experience was that the VSCode extension for Crystal[15] does not (yet) allow for debugging of Crystal apps. This is quite a bit of a pain point, I must admit. But not a big showstopper.

All in all, I've been pleasantly surprised by Crystal: it's a very good idea with a brilliant implementation. It is definitely a potent tool with a great future, and it has a vibrant ecosystem.

[15] https://marketplace.visualstudio.com/items?itemName=crystal-lang-tools.crystal-lang

My First PC

Adrian Kosmaczewski

2022-07-22

I got my first personal computer 30 years ago this month. It was during the summer of 1992; I had just finished my first year of studies at the Collège Sismondi[1], and it was the first summer after we arrived from Argentina[2] that looked like a real holiday.

And one of my greatest childhood wishes, the thing that I wanted so badly since I was a kid, became a reality that summer. A PC. It's weird to think how many doors that computer opened for me. It was a childhood dream come true, and I can still see it as if it were yesterday.

It was a "Microspot" generic desktop tower PC, equipped with an 80386[3] SX CPU running at 8 MHz (it could reach 16 MHz hitting that "turbo" button at the front), with 2 MB of RAM and a 128 MB hard disk drive. It came bundled with a very heavy 13-inch monitor connected to an SVGA video card (no GPUs back then, kids), featuring a maximum resolution of 1024 x 768. I also bought a Sound Blaster 2.0 card[4], a cheap brand-less joystick, and my first ever Logitech Mouse.

My mother bought it for me in July 1992 in an Interdiscount[5] shop at the Rue de Coutance in Geneva, and it must have cost around 3000 Swiss Francs; I don't remember the exact price, but it was definitely around that value. When I look backward, I realize that I knew very, very little about computers; at that moment in time, I could have bought a 486 with Windows 3.1 pre-installed for just a bit more from a different vendor. But, anyway. That's how one learns.

In terms of software, the thing came bundled with MS-DOS 5[6] preinstalled. I bought a separate copy of Windows 3.1[7] (en Français, s'il vous plaît) in September.

What did I do with this first computer? Let's go back down memory lane:

[1] https://en.wikipedia.org/wiki/Coll%C3%A8ge_Sismondi
[2] /blog/thirty-years/
[3] https://en.wikipedia.org/wiki/I386
[4] https://en.wikipedia.org/wiki/Sound_Blaster#Sound_Blaster_2.0,_CT1350
[5] https://www.interdiscount.ch/fr
[6] https://www.youtube.com/watch?v=NfpYrem94q0
[7] https://en.wikipedia.org/wiki/Windows_3.1x

- I tweaked its `config.sys` and its `autoexec.bat` to the extreme, mostly following the instructions I read in PC Magazine and other similar publications.
- I learned to program in QBasic[8] and later in Turbo Pascal[9]. I also typed the occasional listing in assembler provided in one of those computer magazines I read.
- I played games, such as Aces of the Pacific[10] or SimCity[11].
- I wrote school reports using Windows 3.1 Write[12]. I know, the crappiest text editor according to today's standards, but that's what you got with Windows 3.1, and at the time it seemed to me a fantastic tool.
- I bought a copy of MS Access 1.0[13] at the end of 1992, to learn about relational databases.
- Another thing I bought was Lotus Improv[14] for Windows; seriously, my favorite spreadsheet ever.
- And I got into the shareware game. It was quite cool; there was this French mail catalog, updated monthly, with lots of shareware software produced in the US and Europe. There were apps and games of all kinds, and you only paid for the diskettes. I tried plenty of software this way and bought quite a few titles. I wasn't aware of the concept of computer viruses back then. Good times.

I later upgraded this first machine to 4 MB RAM in 1994, and finally sold it to a friend in 1995.

My next machine was yet another generic desktop PC, this one with an 80486[15] DX 2 CPU running at 50 MHz, 8 MB of RAM, capable of running Windows 3.1 and OS/2 Warp 3[16]. I liked OS/2 Warp, it was rock solid but there wasn't a lot of software or hardware made for it.

This 486 was my main PC until 1997, and it was the first one that I connected to the Internet from home, and where I wrote my first web pages. I upgraded this second PC later on with 8 more megabytes of RAM, an internal CD-ROM drive, and even an external Iomega Zip Drive[17] that connected to the printer port. 128 MB diskettes, whaaaat?!

I also sold this second PC to another friend, right before I flew back to Argentina in January 1998.

What about printers? My first was a black & white HP DeskJet 500[18]

[8] https://en.wikipedia.org/wiki/QBasic
[9] https://en.wikipedia.org/wiki/Turbo_Pascal
[10] https://en.wikipedia.org/wiki/Aces_of_the_Pacific
[11] https://en.wikipedia.org/wiki/SimCity_(1989_video_game)
[12] https://en.wikipedia.org/wiki/Microsoft_Write
[13] https://en.wikipedia.org/wiki/Microsoft_Access
[14] https://en.wikipedia.org/wiki/Lotus_Improv
[15] https://en.wikipedia.org/wiki/I486
[16] https://www.os2world.com/wiki/index.php/IBM_OS/2_Warp_3
[17] https://en.wikipedia.org/wiki/Zip_drive
[18] https://www.hpmuseum.net/display_item.php?hw=314

bought in 1992, replaced in 1995 with a color HP DeskJet 560C[19]. I have remained faithful to HP printers ever since, with a small period of Lexmark printers around 2000. I still have an HP printer, an HP Color LaserJet Pro MFP M277dw printer compatible with Linux, Mac, and Windows.

As for the monitor, in 1994 I chipped in and bought a beautiful 14'' Sony Trinitron CRT monitor with the best image I could dream of. It was a fantastic monitor, with astonishingly good image quality. But it was heavy as hell.

It's crazy to think how much things have changed in such a short time, and at the same time, how much they are still the same.

[19] https://www.hpmuseum.net/display_item.php?hw=313

Internet Explorer 4

Adrian Kosmaczewski

2022-07-29

The news recently splashed the demise, disappearance, and final "good riddance" of Internet Explorer. I remembered the first time I encountered the beast. In 1997, Internet Explorer 4.0 was soooooo good compared to anything else, it was hard to believe.

My first contact with the "Internet" was through a gloomy terminal connected to a VAX microcomputer running the VMS operating system. FTP, `talk`, Usenet, and eventually `lynx` were the first impressions I had from the online world. The computer room in my university had a few PCs and Macs scattered here and there, but not all of them were connected to the wider Internet; most were just standalone Windows 3.1 computers meant to run Word and Excel.

It was in one of those few PCs that had a network card where I ran Netscape Navigator 3.0 for the first time. That was around the Summer of 1996, at a time when I felt that I had chosen the wrong carrier (physics). I was so impressed with the web that I ran to the Librairie Ellipse in Geneva to buy Elizabeth Castro's excellent "HTML Visual QuickStart Guide," and following her advice, I got my first website up and running on August 28th, 1996.

The web became an instant obsession for me. I dropped out of college, I left everything for a two-month-long holiday in Argentina. When I came back to Switzerland, I rented an apartment, worked handling luggage in the Aéroport de Genève during the day to pay the rent, and I taught myself JavaScript, Java, and whatever I could learn about web technologies during the night.

By October 1997 I started my first job as a software developer. It all went very, very fast.

One day in September 1997, before I left Switzerland to start my job in Buenos Aires, I went to the said bookstore, Librairie Ellipse, my favorite place to hang out in all of Geneva. Near the cash register, there was a pile of free CD-ROMs, and I picked one.

It had Internet Explorer 4.0. I installed it and fell in love with it. Think about the time when you used Google Chrome for the first time. Same idea.

People like to bash and mock and laugh about Internet Explorer nowa-

days, but the comparison between it and Netscape Communicator 4.0 (the successor of Netscape Navigator) was absolutely in favor of the former. Internet Explorer 4.0 was lightweight, fast (really fast, particularly considering the kind of hardware it had to run on: 386, 486 and Pentium chips), and it was perfectly compatible with Netscape and had lots of great features.

Internet Explorer 4.0 was packed with many features I won't enumerate here, but the most impressive (and the one that arguably had the biggest impact from a historical point of view) was Dynamic HTML. We might laugh at this idea now, but in 1997 being able to manipulate the DOM of a webpage using JavaScript was akin to black magic.

Of course, crafting a website in those pre-jQuery days meant a lot of

```
if (document.all) {
    // yes, this was a way to access any DOM object…
    document.all(…);
} else {
    // This is not IE, remember to use
    // document.layers['…'] to access <LAYER> tags
    // (no <DIV>s in Netscape!)
}
```

Check the archived Netscape documentation[1] if you don't believe me. If you wanted to create complex HTML pages with <DIV> tags, you had to use a server-side rendering technique to change them to <LAYER> objects for Netscape. I'm not kidding.

All things considered, Internet Explorer 4.0 was the browser that crushed Netscape to the ground. It was not a victory by knock-out, it was murder. Two years later, Netscape had virtually disappeared from the market. The thing is, Internet Explorer 4.0 was too good to be true. Thanks to its successors and Microsoft's marketing machine behind, it would take a whole decade for an alternative browser to gain some kind of decent market share. A whole decade. Think about that.

In 1997, I started my career with Internet Explorer 4.0, and yes, it was quite a revelation.

[1] https://web.archive.org/web/19991012215641/developer.netscape.com/docs/manuals/js/client/jsref/index.htm

Microservices or Not? Your Team Has Already Decided

Adrian Kosmaczewski

2022-08-05

Let's take a somewhat tangential approach to the subject of the Microservices architecture. Most discussions about it are centered around technological aspects: which language to choose, how to create the most RESTful services, which service mesh is the most performant, etc.

(Originally published on the VSHN blog[1] in November 2021.)

However at VSHN we have long ago figured out that the most important factor of success for software projects is people. And the thesis of this article is that the choice of Microservices as an architectural pattern has more to do with the organizational structure of your organization, rather than the technological constraints and features of the final deliverable.

Or, put differently: your team has already chosen an architecture for your software, even if they are not fully aware of it. And lo and behold, Microservices might or might not be it.

Definition

First of all we must define the Microservices architecture. What is it?

> "Microservices" is an architectural pattern in which the functionality of the whole system is decomposed into completely independent components, communicating with each other over the network.

The counterpart of the Microservices architecture is what is commonly referred to as the "Monolith", which has been the most common approach for web applications and services in the past 25 years. In a Monolith, all functions of the application, from data management to the UI, are all contained within the same binary or package.

On the opposite side of the street we find the Microservices architecture, where each service is responsible of its own implementation and

[1] https://www.vshn.ch/en/blog/microservices-or-not-your-team-has-already-decided/

data storage.

By definition, Microservices are fine grained, and have a single purpose. They have strong cohesion, that is, the operations they encapsulate have a high degree of relatedness. They are an example of the "Single Responsibility Principle". They are also deployed separately, with visibly bounded contexts, and they require DevOps approaches to their deployment and management, such as automation and CI/CD pipelines.

Very importantly, the Microservices architecture is a "share nothing architecture", in which individual services never share common code through libraries, instead restricting all interaction and communication through the network connecting them.

And last but not least, Microservices (as the name implies) should be as small as possible, have low requirements of memory, and should be able to start and stop in seconds.

Given all of these characteristics, Microservices are, by far, the most complex architectural pattern ever created. It is difficult to plan, it can dramatically increase the complexity of projects, and for some teams, experience has shown that it was impossible to cope with.

History

This idea of "components sending messages to one another" is absolutely not new. Back in 1966, one of the greatest computer scientists of all time, Alan Kay, coined the term "Object Oriented Programming". The industry co-opted and deformed his original definition, which was the following:

> OOP to me means only messaging, local retention and protection and hiding of state-process, and extreme late-binding of all things.
>
> Alan Kay, source[2].

This text is from 2003; the following is from 1998:

> The big idea is "messaging" — that is what the kernal of Smalltalk/Squeak is all about (…) The key in making great and growable systems is much more to design how its modules communicate rather than what their internal properties and behaviors should be.
>
> Alan Kay, source[3].

Alan Kay designed in the 1970s the Smalltalk programming language, based on these concepts. And after reading the texts above, it becomes clear that to a large extent, the Microservices architecture is

[2] http://www.purl.org/stefan_ram/pub/doc_kay_oop_en

[3] https://lists.squeakfoundation.org/pipermail/squeak-dev/1998-October/017019.html

an implementation of Alan Kay's ideas of messaging, decoupling and abstraction, taken to the extreme.

To achieve the current state of Microservices, though, many other breakthroughs were required. During the 2000s, the emergence of XML, the SOAP protocol, and its related web services made the term "Service Oriented Architecture" a very common staple in architectural discussions. The rise of Agile methodologies made the industry pivot towards the REST approach instead of SOAP. During the last decade, the rise of DevOps and the rise of containerization through Docker and Kubernetes finally enabled teams to deploy thousands of containers as a microservice architecture, thanks to the whole catalog of Cloud Native technologies.

Pros and Cons

If Microservice architectures are so complex, why use them? It turns out, they can bring many benefits:

- Since each component can be completely isolated from the others, once teams agree on their interfaces they can be developed, documented, and tested thoroughly, 100% independently from others. This allows teams to move forward in parallel, implementing features that have 0% chance of collision with one another.
- Teams are also encouraged to choose the programming language that fits best the particular task that their microservice must fulfill.
- Since they are by definition "micro" services, they are designed to be quickly launched and dismissed, so that they only intervene when required, making the whole system more efficient and responsive.
- The size of microservices also allows for higher availability, since it is possible to have many of them behind a load balancer; should any instance of a microservice fail, it can be quickly dismissed and a new one instantiated in its place, without loss of availability.
- Systems can be updated progressively, with each team fixing bugs and adding functionality without disturbing the others. As long as interfaces are respected (and eventually versioned) there is no reason for the system to suffer from updates.

But there are many reasons **not** to choose the Microservices architectural approach; among the most important:

- Performance: a system built with Microservices must take into account the latency between those services; and in this respect, Monolithic applications are faster; a function call is always faster than a network call.
- Team maturity: Microservices demand a certain degree of experience and expertise from teams; those new to Microservices

have a higher chance of project failures.
- Cost: creating a Microservices system will be costlier, if anything, because of the overhead created by each individual service.
- Feasibility: sometimes it is simply not possible to use Microservices, for example when dealing with legacy systems.
- Team structure: this is a decisive factor we will talk about extensively later.
- Complexity: it is not uncommon to end up with systems composed of thousands of concurrent services, and this creates challenges that we will also discuss later.

I would like to talk now about the last two points, which are in our experience the biggest issues in Microservice implementations: **team structure** and the **perception and management of complexity.**

Conway's Law

One of the decisive factors that constrain teams' ability to implement microservices is often invisible and overlooked: their own structure. Again, this phenomenon is not new. In 1968, Melvin A. Conway wrote an article for the Datamation magazine called "How Do Committees Invent?" in which the following idea stands out:

> The basic thesis of this article is that organizations which design systems (…) are constrained to produce designs which are copies of the communication structures of these organizations.
>
> Melvin Conway, source[4].

There is extensive evidence, both anecdotal and empirical, of this fact through research.

The corollary of this principle is the following: the choice of a Monolithic versus a Microservices architecture is already ingrained in the very hierarchical chart representing any organization.

One of the services we offer at VSHN tackles, precisely, this very issue. In our "DevOps enablement workshop"[5] we evaluate the degree of agility of organizations, and the extent and improvements that DevOps could bring. Based on that information, we reverse engineer[6] their structure through Conway's Law in order to find a starting point for their digital transformation.

Complex vs. Complicated

Another important point is the distinction of "Complex" versus "Complicated", as these two words can sometimes be confused with one

[4] https://www.melconway.com/Home/Committees_Paper.html
[5] https://www.vshn.ch/en/solutions/devops-enablement/
[6] https://www.vshn.ch/en/blog/reverse-engineering-conways-law/

another in everyday language, and to make things more difficult, the word “Simple” can be used as an antonym of both.

“Complex” is borrowed from the Latin complexus, meaning “made of intertwined elements”. Complexus is itself derived from plectere (“bend, intertwine”). This word has been used since the XVI century to qualify that which is made of heterogenous elements. It shares the same root (plectere) with the medical term “plexus” meaning “interlacing” and used since the 16th century as a medical term for “network of nerves or blood vessels”.

(Source: Dictionnaire historique de la langue française[7] by Alan Rey)

On the other hand, “Complicated” has a similar origin but a different construction: it comes from the Latin complicare, literally meaning “to fold by rolling up”. Figuratively speaking this was taken as close to the notion of embarrassment or awkwardness. The word is composed of the word plicare which means “to fold”. Watches commonly known as “Complications” (such as the Patek Philippe Calibre 89[8], the Franck Muller Aeternitas Mega[9] and the “Référence 57260” de Vacheron Constantin[10]) are, well, complicated machines by definition.

In short, “Complex” and “Complicated” stem from slightly different roots: the Latin root plectere (“to intertwine”) in the former, and plicare (“to fold”) for the latter. Complex conveys the idea of a network of intertwined objects, whose state and behavior are continuously altered by the interaction with their peers in said network. The word complicated implies an intrinsic apparent “obscurity” through folding unto itself, inviting to an “unfolding” discovery process.

Or, put in another way: individual Microservices should not be complicated, but a Microservice architecture is complex by definition. Monoliths, on the other hand, tend to become very complicated as time passes. And of course, neither are simple.

History shows that software developers have a passionate relationship with complication; complicated systems are great to brag about on Hacker News, while maintainers also cry about them in private.

A “best practice” in this context has one and only one basic job: to help engineers translate the complicated into the complex. Most software-related disasters are caused by a simple fact: because of deadlines, organization, tooling, or just plain ignorance, software developers have a tendency to build complicated, instead of complex, systems.

This is another point we take care of in our DevOps Workshop, through the evaluation of the current assets (not only source code, but also

[7] https://fr.wikipedia.org/wiki/Dictionnaire_historique_de_la_langue_fran%C3%A7aise

[8] https://en.wikipedia.org/wiki/Patek_Philippe_Calibre_89

[9] https://www.franckmuller.com/aeternitas/

[10] http://reference57260.vacheron-constantin.com/fr/la-montre-la-plus-compliquee-au-monde

current databases schemas, security requirements, network topologies, deployment procedures and rhythms, etc.)

Migrate or Rewrite? Equilibrium

The complication of Monoliths by itself is not problematic; it makes for tightly bound systems, which tend to be very fast, because as we said, a function call is faster than a network call. After all, we have been building very successful monoliths in the past. But experience shows that they tend to present problems regarding availability and scalability. Microservices represent a diametrally oposed approach, based on complexity rather than complication, but one that solves those issues.

There is a tension, then, between complexity and complication on one side, and organizational forces on the other. Put in other words, there is a tension between monolithic and microservices systems on one side, and more or less hierarchical structures on the other. Achieving equilibrium between these forces is, then, the engineering challenge faced by software architects these days.

Many teams face today the task, either requested internally (from their management) or externally (through customer demand or vendor requirements) of migrating their monoliths into Microservice-based architectures. Architects can thankfully apply a few techniques to find an equilibrium:

1. Start your migration path knowing that very often one does not need to migrate the whole application to Microservices. Some parts can and should remain monolithic, and in particular, proven older systems, even if written in COBOL or older technologies, can still deliver value, and can play a very important role in the success of the transition.
2. Identify components correctly, so that when isolated they will be neither only functionality-driven, nor only data-driven, nor only request-driven, but rather driven by these three factors (functionality, data, and request) at the same time. Pay attention to the organizational chart, and use that as a basis for the decomposition in microservices.
3. Remember that network bandwidth is not infinite. Some interfaces should be “chunky”, while others should be “chatty”. Plan for latency issues from the start.
4. Reduce inter-service communication as much as possible, which can be done in many ways:
    1. Consolidating services together
    2. Consolidating data domains (combining database schemas or using common caches)
    3. Using technologies such as GraphQL[11] to reduce network bandwidth

[11] https://graphql.org/

    4. Using messaging queues, such as RabbitMQ[12].
5. Adopt Microservice-friendly technologies, such as Docker containers, Kubernetes, Knative[13], or Red Hat's OpenShift[14] and Quarkus[15].
6. Implement an automated testing strategy for each and every microservice.
7. Standardize technology stacks around containers & Kubernetes, and create common ground for a true microservice ecosystem within organizations.
8. Automate all of your work as much as possible, knowing that the effort for automation (DevOps, CI/CD) can be amortized over many services, and becomes thus a net investment in the long run.

As mentioned previously, we regularly help organizations in their digital transformation towards microservices, Kubernetes, OpenShift, DevOps, CI/CD, GitLab, and DevOps in general, to support your teams with the tooling they will need in the future. Borrowing Henny Portman's Team Topologies[16] concept, VSHN can support both as an "Enabling Team" (DevOps workshop, consulting) and a "Platform Team" (Kubernetes/OpenShift) to build microservices on, ensuring stability and peace of mind.

Conclusion

Going beyond the hype, Microservice architectures bring great benefits, but can become huge challenges to software teams.

The best way to tackle those challenges consists in reverse engineering Conway's Law[17], and start by the analysis of the human organization of your teams first. Make them independent, agile, and free to choose the best tools for their job. Encourage them to negotiate with one another the interfaces of their respective components.

Let us create and run complex, not complicated, systems. We cannot get away from complexity; that is our job as engineers. But we can get rid of the complicated part.

As a closing thought, I will quote my former colleague and lifetime friend Adam Jones[18], an independent IT consultant from Geneva: in order to achieve success with the Microservice architecture, **you must embed structure in your activities, and remove it from your hierarchy.** It is not about making the structure go away; but instead, moving it to where it does the most good.

[12] https://www.rabbitmq.com/
[13] https://knative.dev/
[14] https://www.redhat.com/en/technologies/cloud-computing/openshift
[15] https://quarkus.io/
[16] https://hennyportman.wordpress.com/2020/05/25/review-team-topologies/
[17] https://www.vshn.ch/en/blog/reverse-engineering-conways-law/
[18] https://twitter.com/cthruair

The Book I Never Wrote

Adrian Kosmaczewski

2022-08-12

I've written and published a few books[1]. But if you look carefully online, you'll find one book with my name on it, even though I haven't written it. This is the story.

If you search for my family name in Goodreads you will find this Apress title: "Pro IOS Table Views: For Iphone, Ipad, and iPod Touch"[2] (yes, with such spelling for "Ipad" and "Iphone", geez), co-authored with Joe D'Andrea, who happens to be an old friend of mine.

But pay attention to the thumbnail; the cover shows a different author! A certain Tim Duckett[3], software developer turned founder and CTO. What happened?

It turns out that Joe and I were contacted by Apress to write this book in early 2010. We started working on the project, and we completed two or three chapters. But Apress was a bit in an internal whirlwind, and they kept changing our project manager; like four or five times in the span of just a few weeks.

That in itself wasn't a problem, but each new person brought "new ideas," a "new direction," or a "new focus" to the project, and after so much back and forth Joe and I respectfully quit. We had lots of work to do (the iPhone market was booming at the time) and we didn't have time for this.

I didn't hear anything else about the project, and in 2012 I discovered that the book had been finally written by somebody else. I ended up meeting the author, Tim, in person at an event in Zürich in 2014. I understand he was working as an iOS developer in Zürich at the time. I remember telling Joe about the anecdote of meeting the actual author of the iOS Table Views book in person!

To makes matters even funnier, the actual book Tim wrote is also on Goodreads[4], and it's hysterical that the page with my name on it has more reviews (albeit with a lower score) than the actual page belonging to Tim's real book.

---

[1] /books/

[2] https://www.goodreads.com/book/show/10706591-pro-ios-table-views

[3] https://www.linkedin.com/in/timduckett/

[4] https://www.goodreads.com/book/show/18626579-pro-ios-table-views

ILOVEYOU

Adrian Kosmaczewski

2022-08-19

Early in the morning of Friday, May 5th, 2000, we were starting yet another day of work at our office in the neighborhood of Olivos[1], north of Buenos Aires, Argentina.

Priorities are different for everyone. In my case, it was catching up with the tech news of the day. For others, it was opening their e-mail.

As I perused some news websites (I wasn't using RSS feeds yet) I read the news of a virulent trojan with catastrophic consequences, making the headlines in Asia and Europe, and as we were waking up, in the Americas too. I learned that it targeted Windows machines (what else?) and that it was written in VBScript[2]. That was the language we were using every day at work.

How did this worm work? The thing would automatically execute when you opened an attachment named `L0VE-LETTER-F0R-Y0U.TXT.vbs` (see the double extension?) and it would immediately overwrite some files of your home directory with copies of itself (those with extensions like JPG, CSS, or MP3), finally sending itself as an attachment to all of your contacts in your address book. Outlook Express[3] and Active Scripting[4] FTW.

It was the (in)famous ILOVEYOU worm[5], also known as CA-2000-04 Love Letter Worm[6].

Precisely as I was reading that article (I swear the timing couldn't have been better) I hear one of my colleagues complain that her computer was not working properly and that all she saw was (and I quote, as I remember it vividly) "it says I love you everywhere!"

Seconds later the coin dropped in my head, jumped to her computer and unplugged its network cable. We then sent an e-mail to all our colleagues worldwide advising them not to open an e-mail with such a title and such an attachment. Thankfully nobody else (that we know

[1] https://en.wikipedia.org/wiki/Olivos%2C_Buenos_Aires
[2] https://en.wikipedia.org/wiki/VBScript
[3] https://en.wikipedia.org/wiki/Outlook_Express
[4] https://en.wikipedia.org/wiki/Active_Scripting
[5] https://en.wikipedia.org/wiki/ILOVEYOU
[6] https://resources.sei.cmu.edu/asset_files/WhitePaper/2000_019_001_496188.pdf

of) had an issue with the worm, even though almost all of us received it in our inboxes.

I kept a copy of the file (which would trigger antivirus alerts for years to come) in some forgotten backup disk. It was so mind-bogglingly simple; start, overwrite the files, open the address book, and send itself to all contacts. That's it. The whole power of ActiveX[7] and COM[8] components, the same programming language we were using in our Windows 2000[9] server-side ASP[10] applications, was used in a completely different, horrendous way.

2000 was the year I started learning about computer security[11]. I started playing with Back Orifice[12] in my free time. At that moment I discovered how fragile software was.

Update, 2023-12-29: Here's a video[13] showing this worm in action.

[7] https://en.wikipedia.org/wiki/ActiveX
[8] https://en.wikipedia.org/wiki/Component_Object_Model
[9] https://en.wikipedia.org/wiki/Windows_2000
[10] https://en.wikipedia.org/wiki/Active_Server_Pages
[11] https://deprogrammaticaipsum.com/the-weakest-link/
[12] https://en.wikipedia.org/wiki/Back_Orifice
[13] https://www.youtube.com/watch?v=ZqkFfF5kAvw

Reusing Apps Between Teams and Environments Through Containers

Adrian Kosmaczewski

2022-08-26

This was my speech for the WeAreDevelopers Container Day[1] on February 3rd, 2021. The talk will feature a live demo showing how to build, optimize, and distribute containers to be reused in as many environments as possible, 100% based on the experience of the VSHN team.

Introduction

Thanks a lot everyone for attending this presentation, and thanks to the WeAreDevelopers team for having me today on this stage. My name is Adrian Kosmaczewski, I am in charge of Developer Relations at VSHN, the DevOps Company located in Zürich, Switzerland.

When organizations start using containers, they usually see them primarily as a lightweight virtual machine. Developers find it easy to learn how to write Dockerfiles, and Operation teams appreciate being able to start and stop them quickly and with low requirements. As time passes, the usual path for those organizations consists in assembling larger applications in Docker Compose, and finally to migrate those apps to Kubernetes.

But this is not the focus of our meeting today. In this talk I will show you how containers are used inside VSHN to not only run cloud native applications, but also to share internal tools in companies

The Challenge of Internal Tools

In the latest "State of Internal Tools" report[2] by Retool,

> More than 80% of respondents said that internal tools are critical to their company's operational success.

However,

[1] https://www.wearedevelopers.com/event/container-day

[2] https://retool.com/blog/state-of-internal-tools-2020/

> Despite how important internal tooling is, teams still struggle to get the buy in they need: more than half of respondents indicated that one of their biggest problems building internal tools is that they don't have enough time or resources to get the job done.

We at VSHN have lots of internal tools[3] and our teams have to juggle between the maintenance of our own customer's systems, which is our core business, and maintaining and keeping updated the myriad tools that we use every and each day. Most of those custom tools are command-line based.

To make things a bit more complex, each VSHNeer can choose their preferred operating system[4], and this means that our technical teams feature users running various Linux distributions (the vast majority, using Ubuntu, Arch, Fedora, and others), Macs, and even Windows PCs.

We also use the best possible programming language for the job; we have internal tools written in Go, C++ , Java, Python, and JavaScript; each has its own requirement, libraries, workflows, and ecosystem.

Go, C and C++ are great languages for multi-platform command-line tools, but to build them, they require a certain degree of knowledge that not all teams might have.

On the other hand, languages like Ruby, Python, and JavaScript have a great library ecosystem, which make them a fantastic choice for prototyping and releasing new features faster. But they require quite a bit of knowledge in terms of libraries, runtimes, and even language versions.

To make matters more complicated, some tools require custom configuration and resources: fonts, styles, images, default configuration files. All of this increases the chances for something to go wrong during installation or runtime.

And to top it off, we need to reuse that knowledge in CI/CD pipelines ; after all, that is one of the core pillars of DevOps, and our slogan is, quite literally, "The DevOps Company". We have to eat our own dogfood.

Our challenge, then, was to find a way to share tools and knowledge with one another and with other systems, with the least possible amount of friction. And for that, we have chosen containers.

Sharing with users in other platforms

The first challenge is then to reuse tools with users in other platforms. To illustrate how containers help us do that, I have created a small TypeScript application. I have chosen to show an example in the

[3] https://handbook.vshn.ch/hb/terminology.html#_tool_naming
[4] https://handbook.vshn.ch/hb/your_first_day.html#_laptop

Node.js / JavaScript ecosystem, because this is arguably the one with the highest headache per minute ratio in the industry.

TypeScript is a language that compiles to JavaScript, which means that we need quite a bit of infrastructure to first compile the application, and then to run it. Lots of dependencies, and lots of "downloading the internet"; we live in the times of npm , which has taken the crown from mvn in the past decade.

The project I'm going to show is called "Greeter"[5]; it is a very simple command-line tool that greets the user on the command line using various characters: Snoopy, R2-D2, a cat, and many more. Characters can "talk" or "think", that is, their text bubbles change depending on the context.

All of this has a certain number of dependencies, and if someone would like to run this app, they would need to install Node.js and perform quite a few operations before being able to run the app.

Our objective today will be reusing this tool in as many environments as possible, with the least amount of stress.

To make things even more interesting, we are going to use Podman[6] instead of Docker[7], and as you can see, I am not using an alias in my command line. This is actually Podman running. And because Docker Hub[8] now imposes certain download restrictions, we are going to use two different container repositories: GitLab and Quay.

As the Joker said, "Gentlemen, let's broaden our minds"[9].

The Dockerfile is using that great technique called "Multi-Stage Builds"[10]. This is probably one of the greatest features Docker introduced in the past 5 years and it has saved countless Petabytes of storage all over the planet.

This technique consists of using a separate image for the build process, and another for runtime. The advantage being that the resulting image that is shared and used is well, as small as possible, and does not contain libraries that are only required during the build.

On the first stage of our Dockerfile[11] we install all the requirements for building our app; that includes the TypeScript compiler, Gulp, and some Gulp plugins. The package.json[12] file of the project shows that there are quite a few dependencies required.

[5] https://gitlab.com/vshn/applications/greeter
[6] https://podman.io/
[7] https://www.docker.com/
[8] https://hub.docker.com/
[9] https://www.youtube.com/watch?v=T8EzcQ0p1Tg
[10] https://docs.docker.com/develop/develop-images/multistage-build/
[11] https://gitlab.com/vshn/applications/greeter/-/blob/master/Dockerfile
[12] https://gitlab.com/vshn/applications/greeter/-/blob/master/package.json

Then we use the pkg[13] project to create a single executable file that contains all the required libraries.

Finally, on the second step of the Dockerfile[14], we use the most lightweight possible distribution (Alpine, of course), we install glibc (sadly, pkg does not yet work very well with musl alone) and finally we copy our executable.

To share our container with others, let's use a public container repository. In this case I will use Red Hat's own Quay.io service, where I create a public project[15] for our "Greeter" image.

We can now push our final container image to Quay.io. To reuse it , I will switch to a VMWare virtual machine where I am running Windows and where I have installed Docker. This is all very "Inception" like, or as Edgar Allan Poe would say[16], "A dream within a dream".

If the Quay.io project was private, I would need to login into my Quay.io account using Docker, of course. But this one is public, so I can pull my newly created image with Docker Desktop for Windows, and boom, I can run it. Very simple. Podman produces images using the same standard as Docker, which means that the interoperability is perfect.

Reusing in GitLab CI/CD

But as you can see I'm hosting my application in GitLab, and GitLab has a lovely CI/CD system and a fantastic container registry, all built in. Let's add a simple .gitlab-ci.yml[17] file in our "Greeter" project, and this way we will be able to store the product of our Dockerfile directly into our project's own image registry[18].

This is even better! Now I can login to `registry.gitlab.com` from Podman or Docker, in any platform, and pull and run this image as I need it.

But this is not all; I would like to reuse my greeter image in other contexts. Take, for example, the case of CI/CD pipelines. We have all faced the hurdles of properly configuring pipelines, be it in Jenkins, TeamCity, GitHub Actions or GitLab. It ain't easy, it ain't pretty, but OMG how cool it is when it works, right?

When it comes to CI/CD pipelines, we at VSHN are great fans of not adding much complexity in the pipelines themselves; we like to be able to reproduce locally the steps performed by the pipelines on our own machines. This makes it much simpler to diagnose and troubleshoot problems, and this is where containers shine.

[13] https://github.com/vercel/pkg
[14] https://gitlab.com/vshn/applications/greeter/-/blob/master/Dockerfile#L28
[15] https://quay.io/repository/akosma/greeter
[16] https://www.poetryfoundation.org/poems/52829/a-dream-within-a-dream
[17] https://gitlab.com/vshn/applications/greeter/-/blob/master/.gitlab-ci.yml
[18] https://gitlab.com/vshn/applications/greeter/container_registry/1669520

To show an example of this at work, here's a little project[19] called "Fortune"[20]; it is a Flask application that returns a random "fortune cookie of the day" per HTTP. This app is built with Python, which has nothing to do with JavaScript (and thankfully so!)

We are going to reuse our beloved "multi-stage build" procedure, this time introducing a new step in the Dockerfile[21] . Let us commit and push this change, and now our CI/CD pipeline run[22] displays an image of Snoopy greeting us directly from the pipeline. Isn't this lovely?

In a few steps, we have taken an application with lots of dependencies, and have bundled it in a format that is suitable for collaboration and reuse in as many systems as we could think of.

Task Examples

What are the kind of things you can do with this approach? At VSHN we have created container images that allow us to do the following things:

- Run tasks on code:
  - Linter
  - Black-box testing
- Run tasks on documentation:
  - HTML dead link verification
    - https://github.com/wjdp/htmltest
  - Style checks using vale
    - https://github.com/vshn/vale
  - Spell checking
    - https://github.com/vshn/hunspell
  - Create PDF or EPUB formats
    - https://github.com/vshn/asciidoctor-pdf
    - https://github.com/vshn/asciidoctor-epub3
  - Creating the index for a search engine
    - https://github.com/vshn/antora-indexer-cli/
  - Live preview of documentation
    - https://github.com/vshn/antora-preview

Another nice benefit is that users can run multiple versions of the same tool, without conflicts! The trick for that consists in always tagging your images properly, and never using `latest` as a tag.

[19] Unfortunately the project does not exist anymore at the original location. Will restore it as soon as I find it.

[20] https://gitlab.com/akosma/fortune

[21] https://gitlab.com/akosma/fortune/-/blob/master/Dockerfile

[22] https://gitlab.com/akosma/fortune/-/jobs/995577647#L131

Gotchas & Drawbacks

As with any technique or approach, there are some things to keep in mind; remember that there is No Silver Bullet[23].

Regarding GitLab

- If you are using GitLab 2FA (and you should), you need to go to GitLab's "Settings / Access Tokens" page and create one of those (with read / write access to the registry, as needed) as the password.

Regarding creating command-line tools

- Follow the Command Line Interface Guidelines[24] and The Art of Unix Programming[25] for your tools; make sure to add help, clear parameters, and to report errors to stderr and to return proper error codes.
  - GitLab CI/CD pipelines AND `docker build` stop executing when the return of any step in the process returns anything else than zero!
    * Use this to your advantage.
    * Make build stop when linter / tests / etc don't pass.
  - If the tool is complex, consider creating a `man` page for it; you can make one very simply using Asciidoctor[26].
- To have some sort of standardization of your internal tools, think about using `cookiecutter` to help you get started.
  - `cookiecutter` https://gitlab.com/akosma/typescript-cli-cookiecutter
- Using `docker run` is a bit more cumbersome than using an executable.
  - Use aliases and Makefiles to make your life easier!
  - Standardize Makefile tasks across projects
    * `make build`
    * `make release`
    * `make clean`
    * `make lint`
    * `make check`
    * `make preview`
- Pay attention to the path of the tool inside the image.
  - This needs to be documented. Inside the container, the application lives in a different filesystem location than what you'd expect.
  - Document this clearly in the README of the tool: provide usage examples!

[23] https://en.wikipedia.org/wiki/No_Silver_Bullet
[24] https://clig.dev/
[25] https://en.wikipedia.org/wiki/The_Art_of_Unix_Programming
[26] https://docs.asciidoctor.org/asciidoctor/latest/manpage-backend/

  - It can get more complicated when using the `--volume` parameter, but it is also very flexible. Remember to add the `:ro` parameter at the end of the volume definition if you just want to read on a directory.
- If your tool generates files, pay attention to the USER that is running your code. In general, remember to `docker run --user "$(id -u)"` so that the files that you generate "belong to you" instead of `root`.
  - Podman does not require it, as it runs not under root , but as the current user.
- If you need to pass secrets to your applications, use environment variables; the easiest way to pass them is using `--env-file secrets.txt`
  - Remember to add the name `secrets.txt` to `.gitignore` ! Don't commit secrets to Git.
  - In GitLab, use the built-in secrets functionality, and create environment variables that you reference in the YAML file
- Use tools like Release It![27] to control your tags and version numbers during build time

Regarding building containers

- Beware of base images that **do not** contain `/bin/bash` and only have `/bin/sh`
  - Either make your shell scripts work with `/bin/sh` if at all possible, or install `bash` in your target image.
  - Similarly, beware of images based in `musl` instead of `glibc` like Alpine! Check out the differences[28] between both.
- Use dive[29] to inspect the internal state of your containers.
- Run `podman ps -a -q` to find out stopped containers.
- Run `podman image prune` to get rid of intermediate images (and save a few GBs of disk space every so often!).
- Use `podman` and `docker` to make sure your containers run in both.
  - At the moment, Podman does not really support Windows and Mac, even though there are binaries for those systems in the website
- Docker has introduced image pull limits: "Docker Hub is slowly starting to enforce a pull rate limit of 100 pulls per 6 hours for anonymous (unauthenticated) IP-Addresses, and 200 pulls per 6 hours for authenticated non-paying users."
  - Use other container repositories: there's a life after Docker Hub! There are self-hosted options, such as: kraken[30],

[27] https://github.com/release-it/release-it
[28] https://wiki.musl-libc.org/functional-differences-from-glibc.html
[29] https://github.com/wagoodman/dive
[30] https://github.com/uber/kraken

Harbor[31], Docker distribution[32], and Sonatype Nexus[33]. And here's a UI[34] for that private registry. There are other SaaS registries than Docker Hub: Docker Pro or Team (paid) plans[35], Quay[36], AWS ECR[37] (Future free public registry announcement[38]), GitHub packages[39] (ghcr.io), and Google Container Registry[40] (gcr.io). And finally, there are registries embedded in OpenShift[41] and the one we saw in GitLab[42].

Tips for Python command-line tools

- Use virtual environments.
- Add the `--no-cache-dir` option to `pip install` in Dockerfiles:

```
$ python3 -m venv .venv
$ source .venv/bin/activate
$ pip install PyYAML
$ pip install PyGithub
$ pip freeze > requirements.txt
$ cat requirements.txt

# later on...
$ pip install -r requirements.txt
```

Tips for tools built with Go

- Since the Go compiler produces self-contained binaries, ready to be used, you can use distroless[43] as the base image, which will reduce the size of your final image drastically.
  - Remember to set the `ENTRYPOINT` properly in your Dockerfile!

Tips for command-line tools built with JavaScript

- Use multi-step builds!

[31] https://goharbor.io/
[32] https://github.com/docker/distribution
[33] https://blog.sonatype.com/nexus-as-a-container-registry
[34] https://joxit.dev/docker-registry-ui/
[35] https://www.docker.com/pricing
[36] https://quay.io/
[37] https://aws.amazon.com/ecr/
[38] https://aws.amazon.com/blogs/containers/advice-for-customers-dealing-with-docker-hub-rate-limits-and-a-coming-soon-announcement/
[39] https://github.com/features/packages
[40] https://cloud.google.com/container-registry/
[41] https://docs.openshift.com/container-platform/4.5/registry/architecture-component-imageregistry.html
[42] https://docs.gitlab.com/ee/user/packages/container_registry/
[43] https://github.com/GoogleContainerTools/distroless

- Run `npm install --prod` to only install the libraries required at runtime.
- Use pkg[44] to create standalone binaries.
- Use the scratch-node[45] base image for minimalistic, small final container images (GitHub[46]).
- Use TypeScript[47] to make your code stronger and easier to manage in your team.

Regarding `CMD` vs `ENTRYPOINT`

Pay attention to the differences between `CMD` vs `ENTRYPOINT`:

> In short, CMD defines default commands and/or parameters for a container. CMD is an instruction that is best to use if you need a default command which users can easily override. If a Dockerfile has multiple CMDs, it only applies the instructions from the last one.
>
> On the other hand, ENTRYPOINT is preferred when you want to define a container with a specific executable. You cannot override an ENTRYPOINT when starting a container unless you add the –entrypoint flag.
>
> Combine ENTRYPOINT with CMD if you need a container with a specified executable and a default parameter that can be modified easily. For example, when containerizing an application use ENTRYPOINT and CMD to set environment-specific variables.

(Source)[48]

Regarding `ADD` vs `COPY`

> …in a nutshell, the major difference is that ADD can do more than `COPY`:
>
> - `ADD` allows `<src>` to be a URL
> - Referring to comments below, the `ADD` documentation[49] states that:
>
> If is a local tar archive in a recognized compression format (identity, gzip, bzip2 or xz) then it is unpacked as a directory. Resources from remote URLs are not decompressed.
>
> Note that the Best practices for writing Dockerfiles[50] sug-

[44] https://github.com/vercel/pkg
[45] https://hub.docker.com/r/astefanutti/scratch-node
[46] https://github.com/astefanutti/scratch-node
[47] https://www.typescriptlang.org/
[48] https://phoenixnap.com/kb/docker-cmd-vs-entrypoint
[49] https://docs.docker.com/engine/reference/builder/#add
[50] https://docs.docker.com/develop/develop-images/dockerfile_best-practices/#add-or-copy

> gests using `COPY` where the magic of `ADD` is not required. Otherwise, you (since you had to look up this answer) are likely to get surprised someday when you mean to copy `keep_this_archive_intact.tar.gz` into your container, but instead, you spray the contents onto your filesystem.

(Source)[51]

Conclusion

Thanks to containers, at VSHN we have reached a very high degree of code reuse. We have reduced the requirements to just one: Docker (or, as you have seen, Podman).

Most of the tools shown in this presentation are open source and free to use. Feel free to build upon them, reuse them, and if you want, to submit your pull requests. We will be very happy to consider your ideas for future versions!

I hope that this presentation has been useful to you, and keep building containers!

[51] https://stackoverflow.com/a/24958548

Best Books of 2014 to 2019

Adrian Kosmaczewski

2022-09-02

In 2007[1], 2008[2], 2009[3], 2010[4], 2011[5], 2012[6], and 2013[7] I published lists of books I enjoyed every year. Starting in 2014 I stopped publishing them every year, but that doesn't mean I didn't keep track of the books I read.

Here's the complete list of books I read and enjoyed from 2014 to 2019. There's a little bit of everything, from poetry to programming to business to philosophy. You can check what I'm reading right now[8] if you're interested, too.

2019

- Structure and Interpretation of Computer Programs[9] by Harold Abelson, Gerald Jay Sussman, and Julie Sussman
- Start with Why[10] by Simon Sinek
- Éloge des mathématiques[11] by Alain Badiou
- L'Être et l'Événement[12] by Alain Badiou
- Dictionnaire amoureux de la philosophie[13] by Luc Ferry
- Lógica aymara y futurología[14] by Iván Guzmán de Rojas
- Expert F# 4.0[15] by Don Syme, Adam Granicz , Antonio Cisternino
- Stylish F#[16] by Kit Eason

[1] /blog/best-books-of-2007
[2] /blog/best-books-of-2008
[3] /blog/best-books-of-2009
[4] /blog/best-books-of-2010
[5] /blog/best-books-of-2011
[6] /blog/best-books-of-2012
[7] /blog/best-books-of-2013
[8] /now/
[9] https://en.wikipedia.org/wiki/Structure_and_Interpretation_of_Computer_Programs
[10] https://simonsinek.com/books/start-with-why/
[11] https://editions.flammarion.com/eloge-des-mathematiques/9782081395930
[12] https://en.wikipedia.org/wiki/L\XeTeXglyph\numexpr\XeTeXcharglyph”0027\relax{}%C3%8Atre_et_l\XeTeXglyph\numexpr\XeTeXcharglyph”0027\relax{}%C3%89v%C3%A9nement
[13] https://www.lisez.com/livre-grand-format/dictionnaire-amoureux-de-la-philosophie/9782259211116
[14] https://searchworks.stanford.edu/view/7825375
[15] https://www.microsoft.com/en-us/research/publication/expert-f-4-0/
[16] https://link.springer.com/book/10.1007/978-1-4842-4000-7

- There's No Such Place As Far Away[17], by Richard Bach
- The Age of Extremes[18], by Eric Hobsbawm
- In the Beginning was the Command Line[19], by Neal Stephenson
- Maus[20] by Art Spiegelman

2018

- Pro C# 7[21] by Andrew Troelsen and Philip Japikse
- Pro TypeScript[22] by Steve Fenton
- F# for Fun and Profit[23] by Scott Wlaschin
- The Feynman Lectures on Physics[24] by Richard Feynman
- The Lean Startup[25] by Eric Ries
- The Year Without Pants[26] by Scott Berkun
- .NET Microservices. Architecture for Containerized .NET Applications[27]
- OOP the Easy Way[28] by Graham Lee
- Principles[29] by Ray Dalio
- Pattern Recognition and Machine Learning[30] by Christopher Bishop

2017

- The Antidote[31] by Oliver Burkeman.
- Working Effectively with Legacy Code[32] by Michael Feathers
- The Positioning Manual for Technical Firms[33] by Philip Morgan
- Blockchain for Dummies[34]
- Code[35] by Charles Petzold

[17] https://www.goodreads.com/book/show/29947.There_s_No_Such_Place_As_Far_Away

[18] https://en.wikipedia.org/wiki/The_Age_of_Extremes

[19] https://en.wikipedia.org/wiki/In_the_Beginning..._Was_the_Command_Line

[20] https://en.wikipedia.org/wiki/Maus

[21] https://link.springer.com/book/10.1007/978-1-4842-3018-3

[22] https://link.springer.com/book/10.1007/978-1-4842-3249-1

[23] https://fsharpforfunandprofit.com/

[24] https://en.wikipedia.org/wiki/The_Feynman_Lectures_on_Physics

[25] http://theleanstartup.com/

[26] https://scottberkun.com/yearwithoutpants/

[27] https://docs.microsoft.com/en-us/dotnet/architecture/microservices/

[28] https://leanpub.com/ooptheeasyway/

[29] https://www.principles.com/

[30] https://www.microsoft.com/en-us/research/publication/pattern-recognition-machine-learning/

[31] https://www.amazon.com/Antidote-Happiness-People-Positive-Thinking/dp/0865478015

[32] https://deprogrammaticaipsum.com/michael-feathers/

[33] https://philipmorganconsulting.com/the-positioning-manual-for-technical-firms/

[34] https://www.amazon.com/Blockchain-Dummies-Computers-Tiana-Laurence/dp/1119365597

[35] http://www.charlespetzold.com/code/

- The Dawn of Software Engineering – From Turing to Dijkstra[36] by Edgar G. Daylight
- John Von Neumann and the Origins of Modern Computing[37] by William Aspray
- 9 Algorithms that Changed the Future[38] by John MacCormick
- Building IBM[39] by Emerson W. Pugh

2016

- The Ghost of my Father[40] by Steve Berkun
- Leonardo Da Vinci[41] by Charles Nicholl
- Inside Out: A Personal History of Pink Floyd[42] by Nick Mason
- Effective Java[43] by Joshua Bloch
- Advanced Swift[44] by Chris Eidhof, Ole Begemann, Florian Kugler, and Ben Cohen
- Learning Android[45] by Marko Gargenta
- Design Patterns for Object-Oriented Software Development[46] by Wolfgang Pree

Also, 2016 was the year when I officially got rid of my Kindle; I converted all those DRM'd books into open formats, organized them in Calibre[47], and got myself a fantastic Kobo Aura H2O[48] reader to enjoy them. You can read more about my ebook strategy in this blog[49].

2015

- Obras Completas[50] by Jorge Luis Borges
- Peopleware 3rd edition[51]
- Agile! The Good, the Hype and the Ugly[52] by Bertrand Meyer

[36] http://dijkstrascry.com/dawn
[37] https://mitpress.mit.edu/books/john-von-neumann-and-origins-modern-computing
[38] http://users.dickinson.edu/~jmac/9algorithms/
[39] https://mitpress.mit.edu/9780262512824/building-ibm/
[40] https://scottberkun.com/the-ghost-of-my-father/
[41] https://www.goodreads.com/book/show/40129.Leonardo_da_Vinci
[42] https://en.wikipedia.org/wiki/Inside_Out:_A_Personal_History_of_Pink_Floyd
[43] https://www.pearson.com/en-us/subject-catalog/p/effective-java/P200000000138/9780134685991
[44] https://www.objc.io/books/advanced-swift/
[45] https://www.oreilly.com/library/view/learning-android/9781449304881/
[46] https://dl.acm.org/doi/abs/10.5555/189440
[47] https://calibre-ebook.com/
[48] https://www.pcmag.com/reviews/kobo-aura-h2o
[49] /blog/sustainable-ebook-strategy/
[50] https://en.wikipedia.org/wiki/Jorge_Luis_Borges_bibliography
[51] http://www.informit.com/store/peopleware-productive-projects-and-teams-9780321934116
[52] https://link.springer.com/book/10.1007/978-3-319-05155-0

- Object-Oriented Software Construction, Second Edition[53] by Bertrand Meyer
- Becoming a Technical Leader[54] by Gerald Weinberg
- Convincing Coworkers[55] by Steve Shogren

2014

- On Grief and Reason[56] by Joseph Brodsky
- North of Boston[57] by Robert Frost
- Wondrous Moment[58] by Alexandre Pushkin
- Final Meeting[59] by Anna Akhmatova
- The Secret Life of Walter Mitty[60] by James Thurber
- The Essays[61] by Arthur Schopenhauer
- The Art of Literature[62] by Arthur Schopenhauer
- NYC Basic Tips and Etiquette[63] by Nathan W. Pyle
- Too Much Happiness[64] by Alice Munro
- El Regreso del Jovel Príncipe[65] by Alejandro Roemmers
- Geek Sublime[66] by Vikram Chandra
  - An absolute surprise, a delightful read. Astonishing and very, very deep. The geek part is actually almost irrelevant. The key is language, communication, gender, history, preservation, unity, rasa and dhvani through literature and poetry. Extraordinary.
- Jony Ive: The Genius Behind Apple's Greatest Products[67] by Leander Kahney
- The Swift Programming Language[68] by Apple
- Using Swift with Cocoa and Objective-C[69] by Apple
- A Swift Kickstart[70] by Daniel Steinberg
  - I actually was one of the reviewers of this book.

[53] https://archive.eiffel.com/doc/oosc/
[54] https://geraldmweinberg.com/Site/Technical_Leader.html
[55] https://leanpub.com/convincingcoworkers
[56] https://us.macmillan.com/books/9780374539061/ongriefandreason
[57] https://en.wikipedia.org/wiki/North_of_Boston
[58] https://www.goodreads.com/book/show/5977679-wondrous-moment
[59] https://www.amazon.com/Final-Meeting-Selected-Poetry-Akhmatova/dp/1438234732
[60] https://en.wikipedia.org/wiki/The_Secret_Life_of_Walter_Mitty
[61] https://www.gutenberg.org/files/10741/10741-h/10741-h.htm
[62] https://www.gutenberg.org/ebooks/10714
[63] https://www.wisebread.com/nyc-basic-tips-and-etiquette
[64] https://en.wikipedia.org/wiki/Too_Much_Happiness
[65] https://elregresodeljovenprincipe.com/
[66] https://www.vikramchandra.com/publications/mirrored-mind-geek-sublime/
[67] https://www.amazon.com/Jony-Ive-Genius-Greatest-Products/dp/159184617X
[68] https://www.apple.com/swift/
[69] https://www.apple.com/swift/
[70] https://editorscut.gumroad.com/l/swift-kickstart

D, or What Go May Have Been

Adrian Kosmaczewski

2022-09-09

In my quest to learn more and more programming languages, I recently dipped my toes into the D Programming Language[1]. My reaction to it involves sadness; on the positive side of things, the language is undeniably brilliant.

On the other side, it is an almost completely ignored language, at least in the circles I use to move. In my 25 years of professional experience, I only met one developer using D for work; the language hardly (if ever) shows up in technical discussions. The official website says[2] that D is used by eBay, Netflix, Facebook, and others, so thankfully it has found certain niches where it shines.

But D deserved more.

As usual, I implemented my Conway[3] and Fortune[4] projects with it. The development experience is brilliant, on par with Go (hence the title of this blog,) C#, TypeScript, and other modern languages. The compiler generates fast code fast, the language has lots of modern features, and in general, it qualifies as a boring language[5]. Again, in my own scale of values this is a positive trait of a programming language.

Developers coming from C++, C#, Java, and Go will find D immediately approachable.

```
import std;
import core.thread;
import conwaylib;

void clrscr()
{
    spawnShell("clear").wait;
}

void main()
```

[1] https://dlang.org/
[2] https://dlang.org/orgs-using-d.html
[3] /blog/polyglot-conway/
[4] /blog/fortune-apps/
[5] /blog/dart-is-boring/

```
{
    Coord[] alive = blinker(Coord(0, 1));
    alive ~= blinker(Coord(0, 1));
    alive ~= beacon(Coord(10, 10));
    alive ~= glider(Coord(4, 5));
    alive ~= block(Coord(1, 10));
    alive ~= block(Coord(18, 3));
    alive ~= tub(Coord(6, 1));

    World world = World(30, alive);
    int generation = 0;
    while (true)
    {
        clrscr();
        generation++;
        writeln(world);
        writeln(format("Generation %d", generation));
        Thread.sleep(dur!("msecs")(500));
        world = world.evolve();
    }
}
```

In many ways, D is like Go. It uses garbage collection, has a nice library with everything you need out-of-the-box, is cross-platform (available for Windows, macOS, all major Linux distributions, Android, and even FreeBSD,) has functional programming features, has integrated unit testing, a package manager, and much more. It also adds some features that Go has not. For example, D integrates unit tests with documentation generation (a fantastic idea!) It also features class inheritance–because, after all, D was meant to replace C++.

Unfortunately, and quite obviously, D is not Go. I say unfortunately because such a language, with such an ecosystem around it, and with such luminaires at its helm (Walter Bright[6] and Andrei Alexandrescu[7] being just a few of them), deserved better. Better, for example, than not even being mentioned in the Stack Overflow 2021 survey[8] of most popular technologies. This is, simply put, not fair; dura lex, sed lex.

And now Google, after Go, Dart, and how many more languages, is positioning yet another one, this time called Carbon[9], as a successor of C++. **Such privilege belongs to D, not to Carbon.** But market forces are something to be reckoned, and at this point it is not obvious how the D Language Foundation[10] could curb them.

Politics and stock market value have a lot to say in the war for programming language supremacy. Go has had its “killer app” with Ku-

[6] https://en.wikipedia.org/wiki/Walter_Bright

[7] https://en.wikipedia.org/wiki/Andrei_Alexandrescu

[8] https://insights.stackoverflow.com/survey/2021#technology-most-popular-technologies

[9] https://9to5google.com/2022/07/19/carbon-programming-language-google-cpp/

[10] https://dlang.org/foundation/about.html

bernetes, and as a result Go is 15th in the August 2022 TIOBE ranking[11], while D appeared 20 positions lower. Go is 11th in the IEEE Spectrum ranking[12], while D appears at the 27th position. Go is 9th in PYPL[13] at the time of this writing, and 16th in the RedMonk ranking[14], and D doesn't even appear in either list.

D is ready, powerful, easy to learn, compatible, cross-platform, fast, extensible, and fun. There's a lot to like in there. But it lacked a "killer app."[15]

[11] https://www.tiobe.com/tiobe-index/
[12] https://spectrum.ieee.org/top-programming-languages-2022
[13] https://pypl.github.io/PYPL.html
[14] https://redmonk.com/sogrady/2022/03/28/language-rankings-1-22/
[15] https://en.wikipedia.org/wiki/Killer_application

The Languages That Bend Your Mind

Adrian Kosmaczewski

2022-09-16

Many programming languages have been sold to unsuspecting software developers with enticing descriptions, promising "transformative experiences" that "irrevocably alter their way of thinking" and other ethereal descriptions, seemingly belonging to other categories of products, such as yoga classes, religions, drugs, progressive rock albums, or role playing games.

Most of the languages advertised as such belong to the "functional" category; a moniker usually justified by their lack of "side effects," which for most purists is a bad, bad thing.

The first one in the list of mind-bending languages is of course Lisp[1], carried on by its quintessential idea; that the language looks exactly like its primary data structure, and thus, as such, naturally lends itself to metaprogramming. The name of this wonderful property is "Homoiconic[2]." It is also the first language implementing Alonzo Church's lambda calculus. Not a small feat. Richard Stallman, Robert Morris, and Paul Graham are the great apostles of this religion, spreading the word of the church, because it's not just lisp service. Sorry, I had to write that one.

APL[3] is the next in the list. A language that consists of a mathematical notation able to parse and process vast amounts of data with the shortest of syntaxes, for which you need a special keyboard. A small but important community keeps this language alive, touting its capacities and eager to build the most complex programs with the least number of lines of code (usually just one,) maintainability be damned. It awkwardly[4] reminds me of another language.

Then followed Smalltalk[5]. The actual runtime where everything is an object, where everything is inspectable. Its most popular spin-

[1] https://en.wikipedia.org/wiki/Lisp_(programming_language)

[2] https://www.expressionsofchange.org/dont-say-homoiconic/

[3] https://en.wikipedia.org/wiki/APL_(programming_language)

[4] https://en.wikipedia.org/wiki/AWK

[5] https://deprogrammaticaipsum.com/the-absolute-no-frills-quite-ignorant-very-incomplete-and-certainly-flawed-beginners-guide-to-smalltalk/

off, called Ruby[6], took the crown thanks to Matz[7], _why[8], and DHH[9] in the 2000s, spearheading the birth of a large number of Web 2.0 successes: Twitter, GitHub, GitLab, Airbnb, Dribbble, Shopify, and many more. Alas, with a somewhat lackluster performance and fail whales[10], a problem that Crystal[11] and Elixir[12] are still trying to solve.

Another recent mind bending language (although it took 20 years to reach that status) is Haskell[13]. Miran Lipovača's book[14] clearly played a role in this case. The timing of this book (2009) couldn't have been better, given that many languages released during the 2010's claimed Haskell as one of their major inspirations. Another illustrious member of the functional programming family.

We could also add F#, who as the bread-winning member of the ML family gets a lot of (deserved) press for its usefulness and expression power. Apart from Don Syme, the creator of F#, arguably Scott Wlaschin[15] is the one stretching F# the most.

Using any of these languages, claim their pundits, is equivalent to a good dose of LSD, coupled with the vision of a thousand gods on top of a pyramid singing Dr. Alban's hit "Sing Hallelujah!"[16]

Some of these mind-bending languages have had notorious uses in finance, for example: most notably Smalltalk and F#. Lisp had its commercial hour of glory with Viaweb[17].

Of course, neither COBOL[18], nor Fortran[19], nor BASIC[20] and its `goto` statement[21], nor C[22] or C++ (although it sometimes does[23]), nor Java[24] or C#[25], nor Python[26], nor Go[27] or Rust, belong to the "mind-twisting" category. They are impure languages, as impure and as imperfect as the world they try to represent.

---

[6] https://en.wikipedia.org/wiki/Ruby_(programming_language)
[7] https://en.wikipedia.org/wiki/Yukihiro_Matsumoto
[8] https://whytheluckystiff.net/
[9] https://twitter.com/dhh
[10] https://www.theatlantic.com/technology/archive/2015/01/the-story-behind-twitters-fail-whale/384313/
[11] /blog/crystal-is-a-surprise/
[12] /blog/elixir-and-phoenix-framework/
[13] https://en.wikipedia.org/wiki/Haskell
[14] http://www.learnyouahaskell.com/
[15] https://scottwlaschin.com/
[16] https://www.youtube.com/watch?v=YRqBcDwG8vs
[17] https://en.wikipedia.org/wiki/Viaweb
[18] /blog/hay-un-lenguaje-llamado-cobol/
[19] https://en.wikipedia.org/wiki/Fortran
[20] /blog/basic-standards/
[21] https://homepages.cwi.nl/~storm/teaching/reader/Dijkstra68.pdf
[22] /blog/pjsip-snippets/
[23] /blog/blow-your-mind/
[24] /blog/not-exactly-what-i-meant/
[25] /blog/quick-comparison-of-c-sharp-and-ruby/
[26] /blog/my-first-django-project/
[27] /blog/thoughts-about-googles-go-programming-language/

Those are the languages everyone complains about[28], as Stroustrup once said. The useful languages[29], as categorized by Perlis. The boring ones[30], as I call them. But maybe, just maybe, that's what explains their success in our industry and their simpler approachability; they embrace side effects and imperfection as natural features of our existence.

In any case, it'd be nice if at some point, as an industry, we could grow past this language fetish and move on.

[28] https://www.goodreads.com/quotes/226225-there-are-only-two-kinds-of-languages-the-ones-people

[29] https://deprogrammaticaipsum.com/alan-perlis-and-the-evolution-of-programming-languages/

[30] /tags/boring-tech/

How to Use a Microphone

Adrian Kosmaczewski

2022-09-23

At some point in your professional speaking life you will have to hold a dynamic microphone[1] in your hands in front of an audience, just like a rock star. Not only should you be aware that it's not a pepper mill[2], there's a few more things to keep in mind while using them.

1. Keep the mic close to your mouth at all times. Because that's where you get the best sound for the audience. Not at 20 cm, not at 10 cm, not next to your hip, not in your pocket, not anywhere, but next to your mouth. Because that's where the sound of your voice comes through, unless you're a ventriloquist. The whole point of having a mic in your hand is for the audience to be able to hear you. Otherwise, people in the audience won't hear you at all and will complain, and you don't want that. Go to rule number 2 for a trick to help you achieve this feat.
2. Rule number 1 is most easily fulfilled if you stick the microphone to your chin, to your lower jaw, right below your lower lip. Keep it there the whole time. Feel it. It's cold and metallic. Now talk and don't even think about moving it away. After a few seconds, it'll be a part of you. This will help you fulfill rule number 1, and the sound of your voice will be loud and clear.
3. Don't wave your hands while talking with a microphone, at least not the hand holding it. This moves the microphone away from your mouth and breaks rule number 1. It also generates a waving sound, whereby the volume of your voice as heard by the audience increases and decreases with a frequency directly proportional to your enthusiasm. The previous sentence was particularly long. Anyway, this is really important for mediterranean folks to keep in mind (I know what I'm talking about.) Remember rule number 1. Use rule number 2 to keep the mic in a single place at all times while you're talking. You can move the other hand if you really have to.
4. Repeat questions on the microphone if the audience don't have their own mic. This is valid with any mic, tbh, not only hand microphones, but it's good to remember this when they cannot fulfill rule number 1 by themselves. If possible, get a separate mic for questions in the audience. There are some funny ones

[1] https://en.wikipedia.org/wiki/Microphone#Dynamic
[2] https://www.youtube.com/watch?v=nLJkPE6x4pI

wrapped in large foam cubes that you can throw around so people can ask their questions. But be careful not to throw them around like baseballs, they can still hurt in case of a direct hit in the face. You want conference attendees to come back next year and not sue you, remember.

5. Make sure the mic is turned on before talking. Just in case, there's usually a red light below that indicates that the mic is on. Then remember rule number 1 and apply rule 2. Your sound engineer can help, if there is one, that is.
6. Don't tap on the mic to check if it's working, if only because the sound it generates can be too loud for some ears, and even worse, you could trigger a feedback loop. If there's a sound engineer, their job is to make the sound of your voice flow through the speakers. It's not your job. Look at them while speaking, and they'll let you know when it's OK with one of those "thumbs up" signs you've probably seen on TV. Re-read rules number 1 and 2 now.
7. Also, don't ask if the audience can hear you. If they don't, they'll let you know by screaming or throwing you something, like tomatoes, rabbits, or pop corn. Remember rules 1 and 2 now.
8. If you feel the need to cough or sneeze on stage, move the mic away from your mouth, preferably behind your back. Not only for sanitary reasons of public knowledge, but also to prevent an explosion of eardrums in your audience. After you've cleaned your mess, go back to rules number 1 and 2. Use hand sanitizer, for the sake of mankind.
9. Don't mic drop[3], even if you're Obama. Microphones are fragile and expensive and your sound engineer will be mad at you. Just follow rules 1 and 2 and you'll do great.

[3] https://www.youtube.com/watch?v=sEp9OUXix-w

Comments about the AWS Serverless Workshop

Adrian Kosmaczewski

2022-09-30

Yesterday I attended the AWS Swiss Cloud Day[1] (what a fantastic event!) and participated in the workshop about Serverless computing taught by a very competent team lead by Magdalena Gargas[2] and Scott Gerring[3].

The workshop consisted in the deployment of a quite extensive Node.js application on top of AWS, using not only Lambda[4] but other services, such as Amplify[5] (where the frontend and backend were installed), Cognito[6] (for user management), DynamoDB[7] (database), Cloud9[8] (web-based IDE), CodeCommit[9] (Git repository), and of course IAM[10] (role-based authorizations).

On the positive side, I was glad to see how all of those services come together to create an application! That was very, very interesting, and I thank the AWS team for that. They were also very helpful with all of us, continuously roaming from laptop to laptop to make sure that everyone's code was up and running, and explaining concepts all over the place. A great job indeed.

On the other hand… I hope not to sound too negative, because the presentation was really interesting. But in general I found that the demo app was too big for just one hour and 45 minutes. I am very well aware that AWS has many moving parts, and that it's extremely complicated to distill the essence of such a platform in a few minutes: I have been (I still am, actually) in the position of writing tutorials for technical audiences, and it's really complicated sometimes.

So instead of useless criticism, here's a list of what I would have done

[1] https://aws.amazon.com/events/swiss-cloud-day/
[2] https://www.linkedin.com/in/magdalenagargas/
[3] https://www.linkedin.com/in/scottgerring/
[4] https://aws.amazon.com/lambda/
[5] https://aws.amazon.com/amplify/
[6] https://aws.amazon.com/cognito/
[7] https://aws.amazon.com/dynamodb/
[8] https://aws.amazon.com/cloud9/
[9] https://aws.amazon.com/codecommit/
[10] https://aws.amazon.com/iam/

differently in such a workshop; of course, I'm no AWS employee nor expert, so please forgive any ignorance on my side.

- I would not have used CodeCommit; I'd simply drop this section. If one can create Lambdas just by pasting code, then let's do that. Cloud9 is needed, though, because of its handy terminal where all the AWS command line tooling is pre-installed.
- Following the previous point, I'd drop the part of the demo consisting in setting up a static website hosted on Amplify, coupled with user account management part in Cognito. Completing this point and the previous took one hour, and I find that it didn't really add anything to the big picture of "Serverless" computing.
- I'd go directly to the gist of the thing: the famous Lambda function, and, personal taste, instead of using JavaScript for it, I'd use Go instead[11]. Go compiles much faster than JavaScript[12], and it's a very readable and concise programming language; besides, it's also the de facto official programming language for the Cloud Native world. The idea is to get a hands-on idea of AWS Lambda as quickly as possible.
- I would have kept DynamoDB just for the Lambda to store data in it; that makes for a useful demo. Read, write, read again, boom, changes!

I always prefer to make demo apps for workshops as simple as possible; in this case, since Serverless is (at least in my mind) somehow synonym of AWS Lambda[13], and if Lambda is all about running snippets of code on demand, I'd just focus on that. Get the code, paste it in the console, and use `curl` to run the request; save and retrieve data. That's it.

From my (admittedly limited) knowledge of AWS, I guess Cloud9 (for its web-based terminal with pre-installed components), Lambda, DynamoDB, and a bit of IAM would have been enough for such a workshop, and all contact with other parts of the infrastructure should be kept to the minimum. In particular IAM: it would of course be really hard not to use it; after all you need to make your Lambda code able to talk to DynamoDB. At the very least.

And if you would like to show a more complex example, you can do so at the end of the session; like a "walkthrough" for attendees, showing how other pieces of the AWS puzzle fit together for more complex scenarios; and you can also make those apps available on GitHub for those interested.

[11] https://docs.aws.amazon.com/lambda/latest/dg/lambda-golang.html

[12] The Node.js application used for this workshop really took a long time to compile; couple that to the time it took to deploy the app to Amplify, and this generated lots of idle times in the workshop.

[13] Or maybe it's just that my perception of Serverless is a bit limited; I give you that. After all there's also this thing called Redshift Serverless[14] in their platform, so maybe there's a lot more to that.

Running akosma software

Adrian Kosmaczewski

2022-10-07

As I celebrate 25 years of work as a software developer, I look back at one of the most thrilling and frantic times in my professional life: those five years in which I worked as a freelance professional.

Yes, I've been around for 25 years. As I said in a previous post[1],

> I started my career as a software developer at precisely 10am, on Monday October 6th, 1997, somewhere in the city of Olivos, just north of Buenos Aires, Argentina. The moment was Unix Epoch 876142800. I had recently celebrated my 24th birthday.

Le sigh. Anyway, I digress.

From 2008 to 2013 I ran my own little software company, mostly dedicated to building mobile apps for Android and iOS, using native and web technologies alike. Being the only person working in it, I was responsible for everything; from writing the code, to managing the business.

[1] /blog/being-a-developer-after-40/

To run the company I used a small set of self-hosted software packages, written in either Ruby on Rails[2] or PHP[3], conveniently hosted at Site5[4].

- I managed all of my projects using my own Redmine[5] instance. It had all the features I needed: projects, tasks, Git and Subversion repository browsing, user management, even Gantt charts, and a lot more. All of my customers had an account in it, and they could open tickets for their applications if needed. Redmine was by far the most important piece of IT infrastructure in my business. It was available at projects.akosma.com[6].
- I ran a second Redmine instance to support my trainings[7][8], available at training.akosma.com[9]. In this website I would create a new project per training session, offering a forum for questions

[2] https://rubyonrails.org/

[3] https://www.php.net/

[4] https://www.site5.com/

[5] https://www.redmine.org/

[6] https://web.archive.org/web/20120112004048/http://projects.akosma.com/

[7] /tags/trainings/

[8] I could have used something like Moodle for that, but I didn't know about it. Redmine did this job very well.

[9] https://web.archive.org/web/20110707142122/http://training.akosma.com/login?back_url=http%3A%2F%2Ftraining.akosma.com%2F

(I offered one year of support after every session), a code download section, a wiki with interesting links, and presentation slides used during the sessions. My students loved connecting there and asking questions.

- I kept track of my contacts, business deals, and more using an instance of Fat Free CRM[10], but because I wanted to be able to update it quickly while on the move and without having to open my laptop, I wrote an iPhone app called Senbei[11] to access it on the go. I could keep track of dozens of customers simultaneously with it, I cannot stress how useful this app was for me. This was hosted in crm.akosma.com, and since it was referenced nowhere, the Internet Archive[12] has no memory of it.
- My main website was running on WordPress[13] on the akosma.com[14] domain (well, at the very beginning[15] it was just a static page.) I always loved WordPress and I most probably always will. The full archives of that blog are available in this website[16] now, converted in Markdown format suitable for Hugo[17] to consume.
- I wrote[18] and hosted my own URL shortener, first as a Ruby on Rails app and then I rewrote it in PHP, called cortito[19]. My friends at Zerofee[20] in London designed a beautiful UI[21] for it. It was first hosted at url.akosma.com[22] and after as akos.ma[23], yes, the same URL that I'm using now for this website.
- I used an instance of MediaWiki[24] as my corporate wiki. Some pages were public, some others not, and I gave access to it to a few customers to work on documents. But I didn't use this app very much; Redmine had per-project wikis that were much more useful. It was running at wiki.akosma.com[25]. By far the most popular page of this wiki was the iPhone URL Schemes[26] page, which received a substantial amount of contributions from other developers, and was referenced in a lot of blogs and Stack Overflow answers back then.
- I had another self-hosted application (maybe at apps.akosma.com or a similar domain, can't remember) that I used to distribute

[10] http://www.fatfreecrm.com/
[11] /blog/senbei-a-fat-free-crm-iphone-client/
[12] https://archive.org/
[13] https://wordpress.org/
[14] https://web.archive.org/web/20110727231551/http://akosma.com/
[15] https://web.archive.org/web/20090715053149/http://akosma.com/
[16] /tags/akosma-software/
[17] /blog/migrating-from-wordpress-to-hugo/
[18] /blog/our-open-source-projects/
[19] https://github.com/akosma/cortito
[20] https://zerofee.org/
[21] /blog/a-nicer-cortito-courtesy-of-zerofee/
[22] https://web.archive.org/web/20091217033857/http://url.akosma.com/
[23] https://web.archive.org/web/20120626114929/http://akos.ma/
[24] https://www.mediawiki.org/wiki/MediaWiki
[25] https://web.archive.org/web/20121117115349/http://wiki.akosma.com/
[26] https://web.archive.org/web/20121111161439/http://wiki.akosma.com/IPhone_URL_Schemes

beta versions of the apps I developed[27] for my customers, including all the required certificates so that iPhones wouldn't choke. This was years before TestFlight[28], people. We had to do this ourselves.

Most of these self-hosted apps stored their data in MySQL databases. I updated them around three or four times a year; which for Ruby on Rails apps is quite cumbersome in general. No containers here (this was before 2014!) Site5 offered an account in a virtual server with Rails, PHP, and MySQL support, so just SSH and have fun.

As for desktop software, I used the following:

- For consulting gigs, I used GrandTotal[29] to generate invoices. The developer of GrandTotal had its own time tracking application back then, called TimeLog[30]. It was a powerful combo, and it helped me a lot. Years after akosma software closed, in another venture, I used Harvest[31] for this purpose.
- I drafted offers and contracts for my customers in Pages[32] with a custom template. I remember that I hired a lawyer specialized in software consulting services to draft the legalese of that standard template for me; one of the best investments I made when I started my business.

I also used a few SaaS services:

- "Google Apps for your Domain" as it was called back then, to host my email, calendar, those kind of things.
- Dropbox for file sharing.
- And I had a paid GitHub account to store some private Git repositories (you needed to pay for that back in the day, now you can get it for free, even if with some limitations[33].)

I also outsourced various services; it's important to outsource those things that aren't your core competency.

- I hired an accountant to do all the required tax filings on my behalf every year. That was also very helpful.
- I hired a >moser[34], a design agency in Lausanne, for the visual identity[35] and stationery design[36].

[27] /tags/apps/
[28] https://testflight.apple.com/
[29] https://www.mediaatelier.com/GrandTotal7/
[30] https://www.grandtotal.biz/TimeLog4/
[31] https://www.getharvest.com/
[32] https://www.apple.com/pages/
[33] https://www.theserverside.com/feature/Want-a-private-GitHub-repository-It-comes-with-a-catch
[34] https://moserdesign.ch/
[35] /blog/new-visual-identity/
[36] /blog/more-about-our-new-graphic-identity/

- I had my akosma shop[37] at CafePress[38] where you could buy coffee mugs, apparel, and even iPhone bumpers with the logo of my company. I know, I know.
- And the lawyer I mentioned a few paragraphs above.

25 years. I still can’t believe it.

[37] https://web.archive.org/web/20110816064936/http://www.cafepress.co.uk/akosma

[38] https://www.cafepress.co.uk/

The Great Idea of Async Work

Adrian Kosmaczewski

2022-10-14

As I mentioned last week[1], I've been in this industry for exactly 25 years. I started my journey as a software developer on Monday, October 6th, 1997. I've had the opportunity of sitting down and writing code for a living for a quarter of a century!

Put it differently, I can say I've been active in this industry for 34% of its existence, if we take 1950 as the "year zero" of computing, or 48% if you consider January 1st 1970, or "epoch,"[2] as the initial point.

One third or one half. In any case, it's quite some time.

Think about this: for how long have there been carpenters, blacksmiths, farmers, fishermen, or merchants, in the world? Well, now think how long 34% of that time would be. Many hundreds of years, certainly, at least for some of those professions. That should give you an idea of how young is our craft.

That brings me to the following idea: we're at the very beginnings of the computer industry. We're still pioneers in this discovery. It is all still very much a "far west." We're still figuring out how to do things with computers. This is all extremely new from the point of view of history.

This is the reason why there are so many new things all time; new frameworks, new languages, new methodologies, new technologies every six months. This rhythm happens because as a species, we're just learning how to deal with computers. We're learning how to build software, day by day. And we're slowly getting better at it every day (well, I'd like to think that!) even if it sometimes doesn't seem so.

This is also why some technologies stay with us, and become what we label as "legacy," like COBOL, for example. I hear some people chuckling in the back, please stay with me.

Now we're hearing about "quantum computing" and whatnot, and at some point we're going to leave the Von Neumann architecture behind[3]. So far, all of the computers in our world have used the Von Neumann architecture. From the Apple Watch in your wrist to the

[1] /blog/running-akosma-software/

[2] https://en.wikipedia.org/wiki/Epoch_(computing)

[3] https://deprogrammaticaipsum.com/a-farewell-to-the-von-neumann-architecture/

controllers on board of the Voyager probes. From those beloved Raspberry Pis to those IBM mainframes running COBOL code every time you swipe a credit card.

Great Ideas

In the past 72 years, we have found some ideas that worked better than others; the Von Neumann architecture was clearly one of them.

COBOL is another. It just works, and it keeps delivering, even if the syntax looks unpalatable to Rust and JavaScript developers today.

Agile is a great idea. It's like we realized that software engineering deals with a substance radically different to concrete, wood, or metal, and that people could actually perform better and create a better thing if we started valuing the things on the left more.

The Model-View-Architecture (MVC) is another great idea, even though React seemed to be replacing it during the past decade, and I'm not entirely convinced it was a good idea, to be honest. But in its own right, MVC was a great idea, and it helped build so many different products in its many incarnations.

In programming languages, we have lots of very recent ideas that have been proven to be fantastic: automatic memory management, for example, either through a garbage collector like in Smalltalk, Java, or .NET, or more recently at compile time, using automatic resource counting like in the case of Objective-C or Swift, or using a borrow checker[4] like Rust's. This has freed software developers from the burden of having to keep track of all those objects on the heap in their heads, and then risking to run out of memory if they didn't, and crashing if they tried to access things they shouldn't be accessing.

Generics is another great idea, allowing developers to make sure that a bag of red marbles only contains red marbles, and neither green marbles nor red tokens. Generics are so much a part of our world that even Go has adopted them lately[5]. They have also brought many benefits: together with enums, Generics brought the idea of `Optional` types to the mainstream, which are making Sir Hoare's billion mistake[6] slowly become a thing of the past. And a similar idea can be used to replace exceptions, thanks to a generic `Result` type. These are all great ideas.

Composition instead of inheritance; here's another great idea that finally has grown in us; when I started programming in object-oriented languages such as Java, inheritance was all the rage. These days we know that it's better to build software in small pieces, each well defined and tested on its own, which we use like Lego bricks to build big-

[4] https://doc.rust-lang.org/beta/rust-by-example/scope/borrow.html

[5] https://go.dev/blog/intro-generics

[6] https://www.infoq.com/presentations/Null-References-The-Billion-Dollar-Mistake-Tony-Hoare/

ger and better software, incrementally. We even reached the point in which neither Go nor Rust have inheritance[7]. Think about that: two of the hottest programming languages today don't feature inheritance, at all.

Type inference[8] is another great idea; it allows your code to have the benefits of a strong type system, but without the verbosity. Your code looks like a scripting language, but everything is strongly typed, and as such it can be optimized and verified in so many interesting ways. And our IDEs can pick up type information as we write our code, and they have become substantially better with time.

Functional programming is now a feature of most languages these days: you can have immutable values in your code, and then `map()`, `filter()` and `reduce()` your data in PHP, C++, C#, Java, Go, Swift, Kotlin, Rust... and of course JavaScript, which was the most misunderstood programming language[9] 20 years ago, and now it's one of the most widely used. Those functional constructs simplify our code, because thanks mapping, filtering and reducing chunks of data, we got rid of lots of `for` and `while` loops, which made code easier to read and maintain.

Unit testing[10] is another great idea; many modern languages integrate a simple unit testing library in the language, so that you could start doing TDD from day one: for example D, Go[11], or Rust (You could. That doesn't mean that people actually do, but they could. They should. I hope you would.) Each one of your components can have its suite of tests, and then you can tell your CI/CD pipeline to execute them for you automatically, and you'll get an e-mail as soon as the build breaks, to make sure that your code is as strong as possible, all the time.

We have also stopped using the `goto` statement, following Dijkstra's timeless advice[12]. It took us 50 years, but we did it, finally. Even if James Coplien says[13] that polymorphism is `goto` on steroids!

Distributed source code management is a great idea; after many different failed attempts at storing source code (Microsoft SourceSafe, anyone?) we went from centralized to distributed, and now Git[14] reigns on top of the category. It has become a standard, and that's a good thing. I personally haven't had to use any other SCM tool than Git for the past 14 years–I still had customers using Subversion and Mercurial around 2008, but they moved to Git ever since.

[7] https://deprogrammaticaipsum.com/the-hype-cycle-of-oop/
[8] https://deprogrammaticaipsum.com/the-truce-of-type-inference/
[9] https://www.crockford.com/javascript/javascript.html
[10] https://deprogrammaticaipsum.com/kent-beck/
[11] /blog/d-or-what-go-may-have-been/
[12] https://homepages.cwi.nl/~storm/teaching/reader/Dijkstra68.pdf
[13] https://deprogrammaticaipsum.com/james-coplien/
[14] /blog/git-for-non-technical-readers/

Containers are another great idea, and Kubernetes[15] is also here to stay; it offers a very flexible environment to run apps. Thanks to containers we don't hear anymore the famous "it works in my machine" excuse; we literally ship your machine. A simple `docker run` or `podman run` will make code, including all dependencies, to download and run in just one operation. I cannot stress how revolutionary and unprecedented this is; I actually expect this pattern to spread to other environments, in particular smartphones and desktop computing.

We have learnt how to deal with legacy code, and that's a great idea; we have lots of good advice and literature that help us as engineers to encapsulate, refactor, test, and talk to legacy code every day. This is knowledge that has only been recently gathered and filtered, and we now have that knowledge at our disposal, and that's a great idea.

Not so great ideas

Pay attention to the fact that I have not mentioned Machine Learning in this list; I do not believe it to be one of the greatest trends of our time, and I believe that there will be yet another AI winter[16] coming up soon.

The biggest issue I have with ML is that we're not eager to see our own biases face-to-face. And ML does precisely that: it slaps our faces with our own biases. Hence, we are appalled at our own racism, our own exclusionary practices; we do not want to tackle those issues, and as such, the practice and uses of ML will get stuck.

And don't get me started on blockchain, web3, NFTs or any of that. At the current stage, it's an unregulated scam of gargantuan proportions, and even worse, a colossal ecological disaster in terms of negligible delivered value by megawatt of energy spent.

Asynchronous Work

And probably the greatest of all great ideas, Open Source[17], has stuck and is here to stay.

Why do I think it is the greatest of ideas? Because it is based around the notion of collaboration and sharing. What we're discovering through software is a new form of social organization. Which is kind of ironic, when you think about it. As practitioners, we're not particularly known for our social skills, are we?

Yet, I am seeing a huge new trend, one that is closely associated to the Open Source movement, and that will have incredible impact in the years to come.

[15] /blog/kubernetes-for-non-technical-readers/
[16] https://deprogrammaticaipsum.com/open-letter-to-a-future-ai/
[17] https://deprogrammaticaipsum.com/open-always-wins/

We have just went through the third summer with a pandemic around us. The virus is still there, but you get the idea. As we get out of our homes, some of us might miss those times working from home. It certainly had some advantages.

Many businesses have been very quick to scrap the "work from home" policies they were forced to setup during the pandemic. For many reasons; in some cases, rightfully so, for there are jobs that cannot be done remotely.

But there are a lot of jobs that can be done from anywhere in the world nowadays, among which software engineering; why don't those jobs stay remote?

There are (at least) two reasons for this. The first one is hard to change, has very deep roots, and is called **trust**. Some company cultures simply do not have trust built-in, and that's a very difficult thing to change. So I won't talk about this, because there's not a lot you can do to change trust (unless you're the CEO, that is.)

But there is another factor that prevents companies from going into full remote work mode, and this is a factor that each one of you can contribute to change.

And that's **working asynchronously**.

Working asynchronously is a different thing than working remote, although one is eminently compatible with the other, of course.

When then pandemic started, many companies jumped to full remote mode, of course, following the instructions of the government. As days started passing by, however, some companies realized that things were actually working very well. They were more productive, and even morale had improved.

At the same time, we heard the horror stories of countless people around the world struggling with remote work. It became clear to some of us that, without knowing it, there was something we were doing right, that others weren't.

And that was something people now call asynchronous work. Inadvertently, many companies had been working asynchronously the whole time. They just didn't know it was called that.

Writing

Working asynchronously works best when you follow some basic requirements.

The most important thing about asynchronous work is that you must write, write, and write. Decisions must be written down, meeting notes must be written down, specifications must be written down.

If you don't like to write, well, maybe asynchronous work is not for you.

Teams working asynchronously usually adopt the concept of RFCs[18], the same ones that were used to create the various networking protocols we use today, or KEPs[19] (Kubernetes Enhancement Proposals.)

Writing is very important for asynchronous work. When decisions are written, they can be agreed upon; when using GitLab or GitHub to manage those RFCs, people can comment, send pull requests, even raise issues about those documents. For each big project, make sure that you write down requirements, features, and ideas, and do that openly; use Git repositories for that. They work really well.

Having ideas in writing allows people to work on them at any time, following their geographical location or sleeping patterns.

Some people work better in the evening, some in the middle of the night, some are morning people. This means that a nice side effect of asynchronous work is that you are more inclusive in terms of neuropsychological patterns.

Talent is not something that works exclusively from 9 to 5. That's a myth. We all have our rhythms.

Since you're writing things down, another side effect is that you need less meetings. Asynchronous makes it possible to redefine even the standup meeting[20].

You can choose to only have meetings in case of deeply complicated issues that require discussion and brainstorming. Have somebody to always write down notes during them, and make sure they end precisely when the timer stops. Meetings are a very expensive thing in companies, and you should only call for one as a last resort.

When you work asynchronously, everyone is fully responsible of their tasks, outputs, and deliverables. Asynchronous work makes collaboration is a core value in every company.

For engineers, tools like Visual Studio Code's "Live Share"[21] extension are very useful to do pair programming over the network; no need to be next to each other, just collaborate on code with the same tool. You can also use it to write long documents in Asciidoc or Markdown.

Async is great for deep work; engineers appreciate the possibility of focusing their attention in one single issue at a time, without interruptions of any kind, in whichever environment they prefer. Unfortunately, many companies are keen on the concept of open spaces[22], which is one of the worst things that have happened to software engineers.

[18] https://www.ietf.org/standards/rfcs/

[19] https://www.cncf.io/blog/2021/04/12/enhancing-the-kubernetes-enhancements-process/

[20] /blog/the-various-styles-of-standup-meetings/

[21] https://marketplace.visualstudio.com/items?itemName=MS-vsliveshare.vsliveshare

[22] /blog/open-spaces/

Open spaces and software engineers don't work well together. You can't have quality software in open spaces. Async work opens the possibility to engineers to work on problems at their own rhythm, in silence, and without interruptions. I cannot stress how important this point is, particularly now that every company is a software company[23].

Another nice side effect of asynchronous working is that stress levels are much lower. Async work is incompatible with micromanagement, and if people need time to go to the doctor, walk the dog, help their kids with homework, or anything else, they just have to notify their teams, leave, do their thing, and pick up where they left later on. Of course you will have deadlines, but managers should instead count on collaboration to help everyone reach their objectives.

Async begets trust and fidelity. Attrition rates go down, because people enjoy both flexibility and being trusted to do their jobs.

What about the Office?

But what happens to our beautiful office? Well, maybe it's time to downscale it, for example by not renewing the lease of some of the space you're renting. This will bring some economies, of course, while still having a nice office for people to come and work into if they prefer.

Let us be very clear: remote work is not for everyone. Some people truly need and enjoy the daily contact with their peers, and having a physical space for them is paramount. The important thing about async work is that your office becomes yet another remote location for your activities; one that happens to have your colleagues in person, and that's a great bonus.

So, if you can, keep your office; use it for team gatherings, apéros, hackdays, company-wide announcements, and for all those moments (and minds) where collaboration requires physical presence.

Tools

What tools can you use for async work? Markdown[24], Asciidoctor[25], and Antora[26] can be used to create documentation websites, put together with GitLab[27]. Merge Requests, just like GitHub[28] Pull Requests, are great async enablers.

[23] https://www.forbes.com/sites/techonomy/2011/11/30/now-every-company-is-a-software-company/
[24] https://en.wikipedia.org/wiki/Markdown
[25] https://asciidoctor.org/
[26] https://antora.org/
[27] https://about.gitlab.com/
[28] https://github.com/

Confluence[29] is a fantastic tool for async collaboration, and it even allows many team members to collaborate in real time on a same page.

Use a chat system like Slack[30], Mattermost[31], or Rocket Chat[32] for live interaction. But even though chat and Zoom[33] are great for live communication, Discourse[34] works better for longer discussions, questions, and actual async brainstorming sessions.

These tools are 100% cross-platform and browser-based; because it's 2022 and people should use the operating system that they prefer. Many software engineers use Linux laptops; designers prefer MacBooks, and management types are more often than not Windows users. They must all be able to interact with one another. Web-based software is compatible with all of these operating systems and more (BSD, anyone?) so people can work as comfortable as possible.

For async management purposes think about using a web-based ERP, CMS, password management tools, a CRM, a bug tracker like Jira[35], and other tools. Use SaaS software whenever possible, particularly if your organization is small, or you don't have a dedicated IT team.

These tools are an enabler to remote and async writing processes; and in turn, writing is an enabler of async work.

Books

If you are interested in async and remote work, I can recommend two great books about it, both written in 2013:

- Remote[36] by Jason Fried and David Heinemeier Hansson; it tells the story of Basecamp, a SaaS company at the origin of the Ruby on Rails web framework.
- The Year without Pants[37] by Scott Berkun; describing the story of how WordPress works, following the author's incursion in the company for a whole year.

I also recommend you subscribe to The Async Newsletter[38] prepared by Doist[39], the same team that brought us Todoist[40] and Twist[41]. They know a thing or two about async work, too.

[29] https://en.wikipedia.org/wiki/Confluence_(software)
[30] https://slack.com/
[31] https://mattermost.com/
[32] https://www.rocket.chat/
[33] https://zoom.us/
[34] https://www.discourse.org/
[35] https://en.wikipedia.org/wiki/Jira_(software)
[36] https://basecamp.com/books/remote
[37] https://scottberkun.com/yearwithoutpants/
[38] https://async.twist.com/
[39] https://doist.com/
[40] https://todoist.com/
[41] https://twist.com/

Conclusion

Writing is an enabler or asynchronous work.

Open Source is an enabler of asynchronous work.

Asynchronous work is an enabler of remote work.

Async is the actual breakthrough, the actual secret sauce of companies like Basecamp, WordPress, GitLab, and many others. This is the factor that made them successful when working remotely.

There is no point in working remote for the sake of it; working asynchronously makes your company more flexible, resiliant, stable, and agile than ever, and it enables you to go remote if you need to.

I believe that async work is the next big thing in management. Well, not only me, but many others think so[42].

And thus, async work will become the next great idea brought forward by the computer industry.

[42] https://async.twist.com/

Killer Apps

Adrian Kosmaczewski

2022-10-21

The D programming language lacked a "killer app" to break through[1]. Another brilliant language suffered from this situation, objectively deserving a much better fate than the one it had; Smalltalk[2].

Other programming languages have had more luck, and did have their "Killer Apps." These "Killer Apps" have taken various shapes across the ages. Some took the shape of other programming languages, some were operating systems, some consumer products, some were specific people, IDEs, open source projects, or other.

Let's look at some programming languages and their "Killer Apps." In this article we're going to concentrate on programming languages, leaving aside killer apps in other contexts, like for example VisiCalc[3] in the case of the rise of the Personal Computer, horizontal killer apps as described[4] by Joel Spolsky, and other cases. The list below is ordered roughly in chronological order of killer app introduction.

- COBOL: IBM System/360[5] and Mainframes.
- C: Unix.
- BASIC: microcomputers[6].
- Pascal: Turbo Pascal[7].
- ABAP: SAP.
- C++: Graphical User Interfaces (`Button` is a `Widget` and so on.)
- Visual Basic: Microsoft Windows and Microsoft Office.
- Perl: the World Wide Web and CGI scripts.
- Java: Netscape.
- VBScript: Active Server Pages[8].
- Python: NumPy[9] and, thanks to it, the Machine Learning craze of the 2010s.

[1] /blog/d-or-what-go-may-have-been/

[2] https://deprogrammaticaipsum.com/the-absolute-no-frills-quite-ignorant-very-incomplete-and-certainly-flawed-beginners-guide-to-smalltalk/

[3] https://en.wikipedia.org/wiki/VisiCalc

[4] https://www.joelonsoftware.com/2012/01/06/how-trello-is-different/

[5] https://en.wikipedia.org/wiki/IBM_System/360

[6] https://en.wikipedia.org/wiki/Microcomputer

[7] https://en.wikipedia.org/wiki/Turbo_Pascal

[8] https://en.wikipedia.org/wiki/Active_Server_Pages

[9] https://numpy.org/

- JavaScript: first Douglas Crockford[10], then XMLHttpRequest and AJAX, then the V8 engine[11] that begat Node.js, and the rest is history.
- Lisp (well, functional programming in general): JavaScript.
- PHP: LAMP[12] and the Dot-com bubble[13] of 2000.
- C#: .NET.
- Lua: game programming.
- Ruby: Rails[14].
- Scala: Twitter[15] and its Failwhale[16].
- Objective-C: iPhone and App Store.
- Go: Cloud Native tools such as Docker[17], Prometheus[18], Kubernetes[19], and pretty much every project sponsored by the Cloud Native Computing Foundation[20].
- Erlang: WhatsApp[21].
- Elixir: Phoenix Framework[22].
- Haskell: Swift[23].
- TypeScript: Visual Studio Code and lately Deno[24].
- Kotlin: Android Studio[25].
- Dart[26]: Flutter.
- Rust: Linux Kernel (maybe?)

In each of the cases above, the existence of the object or event on the right propelled and boosted the use of the language on the left, and vice-versa. Without its "Killer App," the language would have not thrived and achieved the massive popularity and spread it enjoyed; and without the language, the "Killer App" on the right might not have

[10] /blog/douglas-crockford/

[11] https://en.wikipedia.org/wiki/V8_(JavaScript_engine)

[12] https://en.wikipedia.org/wiki/LAMP_(software_bundle)

[13] https://en.wikipedia.org/wiki/Dot-com_bubble

[14] https://rubyonrails.org/

[15] https://www.artima.com/articles/twitter-on-scala

[16] https://www.theatlantic.com/technology/archive/2015/01/the-story-behind-twitters-fail-whale/384313/

[17] https://www.docker.com/

[18] https://prometheus.io/

[19] https://kubernetes.io/

[20] https://www.cncf.io/

[21] https://www.erlang-solutions.com/blog/20-years-of-open-source-erlang-openerlang-interview-with-anton-lavrik-from-whatsapp/

[22] /blog/elixir-and-phoenix-framework/

[23] https://academy.realm.io/posts/swift-summit-abizer-nasir-lessons-from-haskell/

[24] https://deno.land/

[25] Google decided to get rid of the Eclipse-based Android development toolkit (one of the wisest ideas in the history of software development) and asked JetBrains to provide Android Studio based on their IDE toolkit in 2014. This migration paved the way to JetBrains' own new language Kotlin to thrive, but there is another possible reason: given the tortuous lawsuit between Oracle and Google about the use of Java in Android, it is possible that allowing Kotlin to be the primary language for Android may have saved millions of dollars to Google. Who knows. On the other side of the fence, Google hiring JetBrains also left many iOS developers wishing Apple did the same, ditching Xcode for AppCode. Le sigh.

[26] /blog/dart-is-boring/

existed at all.

Languages that lacked a killer app even though they deserved one: D, Smalltalk, Delphi, Ada (?), PL/I, and maybe Crystal[27] nowaways, too, although it's not too late yet.

Interestingly, Dart and Flutter were conceived together; in a way, they mutually reinforced each other, one being the killer app of the other. This is exactly the same case as C with Unix, C# with .NET, Java with the JRE, or ABAP with SAP. The language and its framework mutually reinforcing one another. There's an interesting corollaire of this situation: if you're creating a programming language, you might want to create with it its "Killer App" and release both at the same time. Stephen O'Grady from Redmonk observed this in 2011[28], and it's spot on: Frameworks Lead Adoption[29].

This reinforcing mechanism once again shows how platforms[30] work, how they trigger and sustain their own growth past a critical threshold, and why the competition finds it so hard to fight against them.

It's also worth noting the explosive nature of a "Killer App" upon appearance; it can upset the dynamics of an established market almost overnight, catching everyone (including critics) completely off-guard, changing the course of (predicted) history in a snap.

Objective-C existed since the mid 1980s; the iPhone App Store appeared in 2008 and by 2013 Objective-C was the top language in the TIOBE rankings. Ruby existed for 15 years in relative obscurity until David Heinemeier Hansson used it to create Rails[31] in 2004. JavaScript had existed in browsers for 7 years until Crockford taught us[32] its true nature.

Programming languages can lay dormant for years or decades, and all of a sudden somebody figures out exactly what to do with it, and the world changes completely.

[27] /blog/crystal-is-a-surprise/

[28] https://redmonk.com/sograady/2011/04/27/frameworks-lead-adoption/

[29] The exception to the rule is probably Objective-C, who in spite of coming hand in hand with those wonders of NeXTSTEP and AppKit and Cocoa, had to wait for the iPhone to came along to get some popularity. And then Swift swept all of that away. Le sigh, again.

[30] https://deprogrammaticaipsum.com/geoffrey-g-parker-marshall-w-van-alstyne-sangeet-paul-choudary/

[31] https://www.youtube.com/watch?v=Gzj723LkRJY

[32] /blog/douglas-crockford/

Editorial Kapelusz

Adrian Kosmaczewski

2022-10-28

One day, on a rainy Saturday evening in the autumn of 1989, I taught myself derivatives and integrals. Yes, I know, I didn't have many friends back then, talk about peak nerd stuff.

How did I do that? That particular afternoon (the exact date is lost in my memory) I rode bus 161 and went to the permanent book fair at Plaza Italia[1] in Buenos Aires, a place that I loved to visit every so often. I always loved bookstores, and I enjoyed going to Plaza Italia to find interesting used books about science and maths. They were affordable, and mostly in good condition.

It's in that book fair that I found some books about Mathematical Analysis written by the "saint trinity" of Argentine maths: Celina Repetto, Marcela Linskens, and Hilda Fesquet, published by Editorial Kapelusz[2][3], one of the most important textbook editors of Argentina in the second half of the twentieth century.

Professors Repetto, Linskens, and Fesquet published (together and separately) the best math textbooks ever written in all of Argentina's history. They featured clear explanations, lots of exercises with their solutions, and they were perfect for self-learning. A quick image search[4] gives an idea of the large catalog of math textbooks these ladies co-authored: Algebra, Geometry, Arithmetics, Analysis, Trigonometry… All gems, excellent textbooks. Maybe not comparable to those published by Mir[5] in the Soviet Union roughly at the same time, and which these three ladies most probably knew about, but excellent textbooks for high school students in any case.

I learnt a few months ago that my father studied maths in the class of professor Repetto during his time studying Architecture in the Universidad de Buenos Aires, around 1965. According to my father, she

[1] https://en.wikipedia.org/wiki/Plaza_Italia%2C_Buenos_Aires

[2] https://www.editorialkapelusz.com/

[3] Editorial Kapelusz was usually pitched against Editorial Estrada, because that's a thing we do a lot in Argentina: dichotomies, counterpoints, and contrapositions. Rosas versus Sarmiento, agriculture versus industry, River versus Boca, Clemente versus Muñoz. Argentines will understand.

[4] https://duckduckgo.com/?q=Repetto+Linskens+Fesquet+kapelusz&t=ffab&iax=images&ia=images

[5] https://mirtitles.org/category/mathematics/

was a tough but brilliant teacher. For my part, I studied maths in high school with a disciple of professor Repetto: professor Elisa Quastler, another brilliant teacher who translated from Russian to Spanish at least one book, the “Introduction to The Theory of Groups” by Alexandroff (a book that is available in English at the Internet Archive[6], by the way.)

And that’s how, during a rainy Saturday evening, I learnt how to calculate derivatives and integrals of continuous, simple, one-variable functions. Not that I became an expert in the subject, but I got a very good idea of the methodology, to such an extent that when I passed my exams of Maturité in the Collège Sismondi[7] of Geneva in 1993, I could ace all those math exercises and more.

So here’s my belated thank you to the three great ladies of Argie maths, because they helped me get the best grades through all those years. Most importantly, they tried to make Argentina a better place, the one it deserved to be but wasn’t[8].

Update, 2023-04-14: Somehow I forgot to mention the fact that Charly García immortalized the “Manual de Kapelusz” (ever-present in Argentine schools back in the 80s) in his 1984 song “No Se Va A Llamar Mi Amor”[9] from the outstanding album “Piano Bar”[10].

[6] https://archive.org/details/alexandroff-an-introduction-to-the-theory-of-groups

[7] https://en.wikipedia.org/wiki/Coll%C3%A8ge_Sismondi

[8] /blog/the-argentine-brain-drain/

[9] https://open.spotify.com/track/5HWfFv6sEBnpfRgJYOSCPv?si=6e4c543edc4746f6

[10] https://open.spotify.com/album/17utekM9a95MchXbkbh47k?si=SRqrPzElTXineq2r_pJH-Q

Mastodon

Adrian Kosmaczewski

2022-11-04

Seeing Twitter becoming the Muskverse was the last straw that took me to revive the Mastodon account[1] I opened in 2017 and leave Twitter for good.

So there it goes: please follow me now at @akosma@mastodon.online[2]; some of you might have seen that the Twitter logo is gone from the sidebar of this website (or footer if you read this through your mobile) so the change is happening for real. Thankfully most of the people I regularly follow on Twitter are already on Mastodon, and lots more are coming in every day.

Mastodon takes a bit to get used to, but it is simply healthier and refreshing to be in such a platform again; it feels like Twitter in 2007. Let's see how long we can keep it that way.

I also created a Mastodon account for De Programmatica Ipsum, and you're very welcome to follow it as well: @deprogrammaticaipsum@mas.to[3].

See you in the Fediverse[4] instead!

[1] https://mastodon.online/@akosma
[2] https://mastodon.online/@akosma
[3] https://mas.to/@deprogrammaticaipsum
[4] https://en.wikipedia.org/wiki/Fediverse

Containers and DLL Hell

Adrian Kosmaczewski

2022-11-11

Back in the 1990s, shared libraries were all the rage. Instead of having to ship a 20 MB `*.exe` file to your customer in various floppy disks, you could cut some code out, put it in a set of `*.dll` files, and reuse that code across all your products. Every vendor would then install lots of DLL files in your system, and they would be reused by other apps from the same vendor.

Why did they do that? Well, storage space was expensive back then. An internal 1 GB hard disk in 1995 costed me around a thousand Swiss Francs, and it filled fast with all those apps (well, I also had Windows 3.1 and OS/2 Warp installed simultaneously.) So reusing code in DLLs appeared, at least at first sight, as a valuable mechanism to reduce bloat for software vendors, and to avoid repeating themselves.

The truth is that those DLLs multiplied themselves like bacteria, and led to a concept that made headlines back in 1995: DLL Hell. The concept was so pervasive and problematic that it got its own Wikipedia entry[1].

Talk about being a celebrity.

Even Microsoft couldn't cope with the DLL explosion; apps would crash at runtime because they would load DLLs with the same name, but with different versions of the same functions inside. It was really a mess.

The most egregious example of this mess was Visual Basic itself, and its suite of `VBRUN300.DLL` libraries, of which it seemed there were gazillion versions in shareware collections everywhere. Which one to use? Well, you're out of luck; try them one by one until your game ~~runs~~ doesn't crash anymore.

The best way to prevent DLL Hell was (still is) to link binaries with libraries at build time. But of course, if your storage is expensive, well, it's not the best way to do things, because the final executable will be more bulky at the end. But, look ma! No DLL loading at runtime! Static binaries are the best.

But this is only valid for compiled languages, really. If your app is written in PHP, Python, Ruby, Perl, Lua, or (more recently) JavaScript,

[1] https://en.wikipedia.org/wiki/DLL_Hell

well, you'll need to install a few gigabytes of packages and runtimes for them to run. `README` is your friend, and then `npm install` or `pip install` or `gem install` until everything works.

We're in 2022 now, almost 30 years later, and statically built binaries have won, and even for scripting languages! They are called containers now; Docker containers, pods, what have you, and they encapsulate[2] not only the final executable and its libraries, but also any other piece of runtime code they might need. Like the whole .NET framework, or the PHP runtime, or a complete version of Ruby and Ruby on Rails[3] including all the gems your app needs to get things done.

It does not matter anymore if your binary is actually statically linked, though; apps written in Ruby, Python, PHP, JavaScript, Lua and other similar languages are just interpreted on the fly by the runtime installed within the same container.

The downside is that with those languages your final container image could easily stretch into the gigabytes… not very convenient if you want to run many copies of those containers in the same load-balanced Kubernetes[4] deployment. Oops.

This all means that compiled languages are making a nice comeback right now; Go[5], Rust[6], D[7], C++[8], .NET[9], Crystal[10], C[11], they can all be used to generate a small self-contained executable, and with it, a very small final container image.

Specify in your `Dockerfile` a base image `FROM scratch`, copy your binary, and your container image is now just 15 MB big. Push it, share it[12], pull it, and reuse it. And apparently .NET 7 (the latest version) includes a new AOT feature[13] that makes really small native binaries. Finally. Oh, and you can even `dotnet publish` directly as a container.

Talk about convenient. Make small container images, people.

[2] /blog/fortune-apps/
[3] /blog/the-technical-news-of-the-day/
[4] /blog/kubernetes-for-non-technical-readers/
[5] /blog/thoughts-about-googles-go-programming-language/
[6] /blog/first-web-app-in-rust/
[7] /blog/d-or-what-go-may-have-been/
[8] /blog/blow-your-mind/
[9] /blog/a-linker-for-joel/
[10] https://akos.ma/blog/crystal-is-a-surprise/
[11] /blog/how-knowing-c-and-c-can-help-you-write-better-iphone-apps-part-1/
[12] /blog/reusing-apps-between-teams-and-environments-through-containers/
[13] https://learn.microsoft.com/en-us/dotnet/core/deploying/native-aot/

Avoid These Things Because Insecure

Adrian Kosmaczewski

2022-11-18

Avoid all of these things with all your might, beware, don't even think about it. You know it's going to hurt. Stay away. You've been warned. Pay attention.

- Office documents, because Visual Basic macros and ILOVEYOU[1].
- Jira and Confluence, because there's a new CVE every week and it's obvious at this point that Atlassian has thrown the towel.
- Zoom, because it's a bad joke.
- Windows, because please please please tell me I don't need to tell you why.
- macOS, because Apple couldn't care less about fixing vulnerabilities and also `goto fail`[2].
- Linux and other open sores[3] projects, because who knows what's in it, damn communists.
- Android, because store full of malware.
- iOS, because in-app purchase scams.
- E-mail, because phishing.
- Unsigned 32-bit executables you compiled 20 years ago and just found in an old backup, because Windows Defender or whatever sandbox in your operating system will block them from running and good luck compiling them again.
- Web browsers, because (in roughly chronological order) Java applets and ActiveX components and JavaScript `eval()` and Flash players and tracking cookies and privacy issues and phishing and malware and ads and Facebook and…
- Intel chips, because Pentium FDIV[4] and Spectre[5] and Meltdown[6] and Joanna says so[7].
- Docker Hub, because rogue containers.
- Kubernetes, because secrets encoded with base64.
- Consoles and terminals, because you might inadvertently execute `rm -rf /`.

[1] /blog/iloveyou/
[2] https://en.wikipedia.org/wiki/Unreachable_code#goto_fail_bug
[3] https://deprogrammaticaipsum.com/open-always-wins/
[4] https://en.wikipedia.org/wiki/Pentium_FDIV_bug
[5] https://en.wikipedia.org/wiki/Spectre_(security_vulnerability)
[6] https://en.wikipedia.org/wiki/Meltdown_(security_vulnerability)
[7] https://blog.invisiblethings.org/papers/2015/x86_harmful.pdf

- Short passwords, because easy to crack.
- Long passwords, because quantum computers are coming.
- Passwords managers, because… wait, no, this one is OK to use.
- Two-factor authentication over SMS, because SIM card spoofing.
- HTTP, because lack of encryption.
- HTTPS, because man-in-the-middle attacks.
- Telnet, because we're not in 1993.
- FTP, because, ahem.
- USB drives, because they can take over your machine.
- OpenSSL, because Heartbleed[8].
- Electron and React Native apps, because JavaScript.
- JavaScript, because npm.
- Twitter, because their chief of security quit.
- Facebook, because Zuck is more interested in legless avatars.
- Google products, because they'll be phased out soon.
- BASIC, because GOTO considered harmful.
- C and C++, because buffer overruns.
- PHP (and by extension WordPress, MediaWiki, Moodle…), because SQL injection and XSS attacks.
- Cookies, because the European Union says so.
- Package managers, because no SBOMs.
- Chats, because impersonation and dubious encryption standards and on the Internet nobody knows you're a dog.
- Programmers, because shift left is a myth and good luck communicating with them[9].
- Cryptocurrencies, because.

Actually, just stop using computers in general, because they were made by humans, and you know how they are.

[8] https://en.wikipedia.org/wiki/Heartbleed

[9] https://deprogrammaticaipsum.com/the-impossible-dialogue/

How to Use Demo Magic

Adrian Kosmaczewski

2022-11-25

My colleague Tobru[1] recently pointed me to demo-magic[2], and I now seriously wonder how could I ever do a live demo without this. The description of the project says it all:

> A handy shell script that enables you to write repeatable demos in a bash environment.

The idea is as simple as this: instead of having to type commands while doing demos or recording videos, just create a script that will:

1. Write the command for you on the screen;
2. Wait for you to press <ENTER>;
3. Execute the command live for you.

And not only that, but it's written in 100% shell script: just save the file locally, and make sure you have pv[3] installed (`sudo apt install pv` should do the trick on Debian, Ubuntu and similar.)

Then write a script as follows:

```
#!/usr/bin/env bash

# Include the magic
. demo-magic.sh

# Clear the screen before starting
clear

# Print and execute a simple command
pe "echo 'hello world'"

# Wait until the user presses enter
wait

# Print and execute immediately
pei "bat my-demo.sh"
```

[1] https://mstdn.social/@tobru
[2] https://github.com/paxtonhare/demo-magic
[3] https://ivarch.com/programs/pv.shtml

Save the script as a file named `my-demo.sh` (well, any name, really) and just `bash my-demo.sh` to get started. Press <ENTER> to print the command, and press it again to execute it. As simple as possible, and very useful when demoing stuff on a terminal.

But you can do much more with `demo-magic.sh`:

- Personalize the prompt;
- Simulate network connections even if offline (whaaat);
- Run hidden commands, not shown to the public;
- Add timeouts to the `wait` command;
- Get help on your `my-demo.sh` script using the `-h` argument;
- Type your own commands, if needed, using the `cmd` instruction.

Simply put, this is a godsend for all the DevRel people reading this. Couple it with asciinema[4][5] and you're in for a treat.

{{< asciicast src="demo.cast" >}}

Update, 2022–11-26: The movie in this page is embedded using the cljoly/gohugo-asciinema[6] Hugo module created by Clément Joly[7].

[4] https://asciinema.org/

[5] asciinema is another gem worthy of its own blog post, because srsly. It's so awesome I don't know where to start.

[6] https://github.com/cljoly/gohugo-asciinema

[7] https://cj.rs/

Cholila

Adrian Kosmaczewski

2022-12-02

Cholila[1] is a little town in the province of Chubut[2] in the middle of Argentine Patagonia[3] inhabited by around 2000 souls. It is mainly known for its National Barbecue Festival[4] and because Butch Cassidy[5] lived there for a while.

I bet you have never heard the name "Cholila" before reading this article. I first heard it through my friend Hernún[6], who was somehow fascinated with it, maybe around December 2000.

Cholila. The name sounds exotic, remote, and secluded. Well, at least in Spanish, or at least to Argies.

It's my first time writing about this period of my life. It feels weird to remember those days.

```
$ cal 2 2001
   February 2001
Su Mo Tu We Th Fr Sa
             1  2  3
 4  5  6  7  8  9 10
11 12 13 14 15 16 17
18 19 20 21 22 23 24
25 26 27 28
```

One rainy day in February 2001, I think it was the 19th or the 20th, I met a woman from Cholila. Her name doesn't matter, we'll call her X. We started dating, and she moved in with me after a few days.

Move fast and break things, they said.

> Once I had a love and it was a gas
> Soon turned out had a heart of glass

I remember driving one day with Hernún in my car. I had a white first-generation Ford Ka[7] we affectionately called "Gutiérrez" for reasons

[1] https://en.wikipedia.org/wiki/Cholila,_Argentina
[2] https://en.wikipedia.org/wiki/Chubut_Province
[3] https://en.wikipedia.org/wiki/Patagonia
[4] https://es.wikipedia.org/wiki/Cholila#Fiesta_nacional_del_asado
[5] https://en.wikipedia.org/wiki/Butch_Cassidy
[6] https://hernun.com.ar/
[7] https://en.wikipedia.org/wiki/Ford_Ka#First_generation_(BE146;_1996)

that don't matter now. I told him that I had met a girl from Cholila, and he rightfully thought I was joking. After all, it's a small town of 2000 people in a country of 40 million. What are the odds?

It was not a good time. Argentina was breaking apart, I was going through therapy, and I still worked a job that I liked but with people with whom I didn't get along.

But I did get along with her. We clicked instantly.

She gave me a copy of the fantastic book Recuentos para Demián[8] by Jorge Bucay[9]. She wrote a few words in the first page. I still have it.

Times were accelerating. I knew change was near.

```
$ cal 7 2001
     July 2001
Su Mo Tu We Th Fr Sa
 1  2  3  4  5  6  7
 8  9 10 11 12 13 14
15 16 17 18 19 20 21
22 23 24 25 26 27 28
29 30 31
```

One day in July 2001, I woke up in sweat and tears. I had dreamt of riots. She was by my side; she tried to comfort me. I murmured the words, "we must leave."

I had a hard time sleeping anyway. We lived in downtown Buenos Aires, 300 meters from the Obelisco[10], and one could hear massive gunfire every night. The country's social contract had broken long ago, and it was getting worse every day.

I remember my colleagues at work laughing at me when I told them about those gunshots. "They would say it on TV if it was true." I also remember mentioning that I thought the Convertibilidad[11] or peso-to-dollar parity could only last for so long. More laughs. My time in that place was over.

I quit my job on the morning of July 20th, 2001. I sold my car in a "Car One" dealership along the Panamericana[12] highway. The following week I went to the bank, wired my economies to my mother in Switzerland, and closed my account. I kept some cash and later bought Traveler's Cheques in dollars to carry safely with me.

```
$ cal 8 2001
    August 2001
Su Mo Tu We Th Fr Sa
```

[8] https://www.goodreads.com/book/show/34556350-recuentos-para-demi-n
[9] https://en.wikipedia.org/wiki/Jorge_Bucay
[10] https://en.wikipedia.org/wiki/Obelisco_de_Buenos_Aires
[11] https://en.wikipedia.org/wiki/Convertibility_plan
[12] https://es.wikipedia.org/wiki/Autopista_Panamericana

```
          1  2  3  4
 5  6  7  8  9 10 11
12 13 14 15 16 17 18
19 20 21 22 23 24 25
26 27 28 29 30 31
```

We planned to go live to Costa Rica, and we bought plane tickets for September 15th, 2001. (I know what you're thinking.) But before leaving Argentina she wanted to introduce me to her family in Patagonia, so on August 16th we left Buenos Aires.

We took the night bus to Bariloche[13] and then another one to Epuyén[14] along the famous Route 40[15]. From there, we took a cab and drove a long, lonely dirt road until we reached the house of her family, who lived in the proverbial middle of fucking nowhere.

Near their home were lakes, forests, sheep, a river, and a dirt road. A few years later, that house would be visible on Google Earth.

These people lived almost wholly cut off from the world; if it weren't for a DirectTV dish on the roof and a mobile cellphone with a somewhat faint and shitty signal. I settled into a dull routine of chopping wood, taking long walks, and disagreeing with everyone about anything. I'm charming like that. They were as lovely as me.

The location was gorgeous, but it could have been a better stay there. I didn't get along well with X's family, and the situation deteriorated badly after a few months.

Apparently I didn't get along with many people back then.

No, scratch that.

~~Apparently I didn't get along with many people back then.~~ X had changed. I didn't know why, but she didn't act the same in Cholila as she did in Buenos Aires. I would later understand.

> Love is so confusing, there's no peace of mind
> If I fear I'm losing you, it's just no good
> You teasing like you do

A few weeks after we arrived, we woke up and watched the twin towers go down[16] live on CNN.

(Anecdote: I had visited New York in 2000, and did the mandatory tour to the rooftop observation deck of the south tower[17]. I later learnt my mother was confused and thought I was in New York during that 9/11. I guess I couldn't have been further away from NYC that morning.)

[13] https://en.wikipedia.org/wiki/Bariloche

[14] https://en.wikipedia.org/wiki/Epuy%C3%A9n

[15] https://en.wikipedia.org/wiki/National_Route_40_(Argentina)

[16] https://en.wikipedia.org/wiki/September_11_attacks

[17] https://en.wikipedia.org/wiki/2_World_Trade_Center#Original_building_(1973%E2%80%932001)

Bin Laden quite radically had just changed our plans, so we stayed in Patagonia.

```
$ cal 12 2001
   December 2001
Su Mo Tu We Th Fr Sa
                   1
 2  3  4  5  6  7  8
 9 10 11 12 13 14 15
16 17 18 19 20 21 22
23 24 25 26 27 28 29
30 31
```

Surrealism is not enough to describe some situations. The world as I knew it was tearing itself apart on satellite TV, but the forests, the river, and the dirt road wouldn't care less.

Beginning of December we watched the economy minister (rhymes with sinister) Cavallo[18] announce cash withdrawal restrictions in the country.

We finally left Cholila on December 4th; after the cab to Epuyén, we took a bus and slept that night in Bariloche. On the 5th, we crossed the Chilean border on a bus, went through Osorno, and reached Valdivia[19] that evening. We took another night bus, and on the 6th, we got to Santiago de Chile.

1'400 km in 48 hours from Cholila to Santiago de Chile via Epuyén, Bariloche, Villa La Angostura[20], Osorno[21], and Valdivia. Average speed: around 30 km/h.

In Santiago, over coffee, she finally revealed why she behaved differently in her family's house. A horrendous story that I won't share here.

We went for a walk. I remember people selling pirated copies of the recently released Windows XP in the streets of Santiago. I also remember seeing a magazine cover about Apple releasing a new thing called "iPod" and wondering what that was.

But I digress. We decided to take a small break from each other. I got the news that my mum had had eye surgery, and we decided it would be better for me to stay with her for some time. The idea was to meet again in Santiago de Chile in February 2002 and restart things together.

I still had some economies in Traveler's Cheques; I transferred them to her name. On the 7th, I bought an Air France ticket taking off that evening. The return flight was two months later, around February 20th.

[18] https://en.wikipedia.org/wiki/Domingo_Cavallo
[19] https://en.wikipedia.org/wiki/Valdivia
[20] https://en.wikipedia.org/wiki/Villa_La_Angostura
[21] https://en.wikipedia.org/wiki/Osorno,_Chile

Adorable illusion and I cannot hide
I'm the one you're using, please don't push me aside
We could make it cruising, yeah

We split at Santiago's long-distance bus station. She took a bus to Valdivia. I remember her waving at me from the bus. She was crying.

I took a cab to the airport and flew to Paris that evening. From there, I flew to Geneva on the following day. I landed on the freezing evening of December 8th. Another cab took me to my mother's home in Chemin du Champ-d'Anier street, in Petit Saconnex[22].

12'000 km in 24 hours from Santiago de Chile to Geneva via Paris. Average speed: around 500 km/h.

I weighed just 70 kg, my mother barely recognized me, and I fell ill immediately after arriving. I was exhausted and drained.

Later that month, my mother and I watched the December 2001 riots[23] in Buenos Aires, again on CNN[24]. I saw the gunshots being fired live on TV. I saw the peso-to-dollar parity disappear into oblivion.

The world where I had lived for the past four years disappeared almost overnight.

```
$ cal 1 2002
    January 2002
Su Mo Tu We Th Fr Sa
       1  2  3  4  5
 6  7  8  9 10 11 12
13 14 15 16 17 18 19
20 21 22 23 24 25 26
27 28 29 30 31
```

On January 15th, 2002, I spoke to X on the phone for the last time. We argued. Again. Suddenly the call dropped. It happened often because the mobile signal at their location was feeble. So I redialed her number, but this time the voice on the other end told me something different: the number I dialed was invalid. I redialed. Same. Again. Same. Again. Same.

The number was not invalid, and I had not composed it wrongly.

Once I had a love and it was divine
Soon found out I was losing my mind
It seemed like the real thing, but I was so blind
Much o'mistrust, love's gone behind

I must have retried dozens of times until it was late in the Swiss night. I tried again every day for a whole week. The next day I sent her an

[22] https://fr.wikipedia.org/wiki/Le_Petit-Saconnex

[23] https://en.wikipedia.org/wiki/December_2001_riots_in_Argentina

[24] https://web.archive.org/web/20011227145908/cnn.com/2001/WORLD/americas/12/19/argentina.riots/

email. The following day I kept trying. Phone. Same. Email. Same. Again. Same.

Fast forward twenty years later, I never heard from her ever again. Not on the phone, not on email, not anywhere.

```
$ cal 2 2002
   February 2002
Su Mo Tu We Th Fr Sa
                1  2
 3  4  5  6  7  8  9
10 11 12 13 14 15 16
17 18 19 20 21 22 23
24 25 26 27 28
```

The date of the return flight to Santiago arrived. I woke up, and I decided that I would stay in Switzerland. There was nothing left in Argentina for me.

I cried. I'm like that. I have a terrible habit of longing for the wrong people.

By July, I had moved to a new place in Lausanne and started a new job.

That house is still visible on Google Earth.

> Yeah, riding high on love's true bluish light

PS: I don't judge her, nor anyone should. I moved on, and so did she. I just tell this story so that nobody reading it feels the temptation of ghosting[25] anyone, ever.

[25] /blog/radio-silence/

What Objective-C 3.0 Could Have Been

Adrian Kosmaczewski

2022-12-09

There's a parallel universe, with a parallel WWDC 2014, in which instead of Swift, developers got Objective-C 3.0, and this is what it would have looked like. It's the same parallel universe where Russia doesn't annex Crimea, by the way.

To begin with, Objective-C would have made header files entirely optional, and if you didn't provide one, they'd be automatically generated from the implementation file. Just write your `*.m` files and save.

The `id` type is great but let's be honest, it does not help much. It's a big part of what I called the "west coast" compiler mentality a few years ago[1]. Instead, Objective-C 3.0 introduced the new `auto` type indicator, which triggers type inference, to make the compiler verify that the methods you're calling actually exist; and the type of the `auto` receiving variable would be whatever the method returns.

Objective-C 3.0 would have dropped class prefixes for actual namespace support. And since we're at it, it would have introduced custom operators. Developers love syntax sugar.

A package manager. Yes, Objective-C 3.0 would have brought that, of course. Hosted by Apple, available to developers registered in the App Store program as an added value. With security checks and everything, thank you so much.

You know what? Let's be crazy. Objective-C 3.0 would have removed the requirement for **square brackets**. Yes, I've said it. Extending the dot syntax for method calling. And since we're at it, drop the `@` sign for strings, arrays, and dictionaries. Seriously, just drop it. Very few people uses C-strings in iOS apps, seriously, and if you really wanted to use them, surround the code with one of the newly introduced `c {…}` blocks, and that's it. Whatever you put inside is pure C. Oh, and you've got another block called `cpp {…}` for… you guessed it, C++ code. ABI stability FTW.

In the same vein, let's not forget about the new `rust {…}` block that provides interoperability with this new little language from Mozilla. Who knows, it could be even used in an operating system kernel in the future; Rust looks really promising.

[1] /blog/the-developer-guide-to-migrate-across-galaxies/#4-objective-c-aka-coolia

Since we have a `c {…}` block for pure C, how about we drop the * in front of variables? We know that Objective-C objects live on the heap (except for blocks) so just drop the asterisk. Now Objective-C 3.0 really looks like Objective-J[2], well, apart from the square brackets, which are gone.

What else? Inspired by Go or D[3], Objective-C 3.0 would have included built-in unit tests, using the new… `unittest {…}` block. To stop people from arguing about source code formatting, the new `objcfmt` tool would automatically format code, just like Go or Rust would do. KVC would have evolved into something akin to what LINQ provides for C#, providing live queries with compile-time checks, for more complex data manipulation at runtime. And KVO would have evolved from the Observer pattern to a React-enabling technology, built around data streams.

And because all languages end up in a Kubernetes cluster these days, and because Objective-C 3.0 is a language based on messaging[4], it would naturally compile source code into microservices. Yes, that's right; it would compile your application code into several separate containers[5] ready to run on your orchestrator of choice, and to help developers, it would do the same but into a single binary for local debugging. Of course, production builds would be `scratch` or `alpine` containers, small and with as few dependencies as possible.

All of these new Objective-C 3.0 features would have been enabled via ad hoc special compiler arguments; that is, Objective-C 3.0 would have been 100% compatible with Objective-C 2.0 code, with new features enabled by default on new projects. You could start adopting them little by little, one at a time, without breaking your code.

In any case, this parallel Objective-C 3.0 would have kept the dynamic runtime, adding a little bit of static type checking for those who would really want it and need it. Boring[6], but great.

Keeping Cocoa, evolving it, and building on top of it; not just throwing it away.

But, of course; saying any of this (even though I'm not the first[7]) is simply akin to heresy[8]:

> Have you ever felt developers might get a little stigmatized for using Objective-C, or talking about it on social media?
>
> Steve Troughton-Smith: A little? It's deeply unpopular to use ObjC or say you like ObjC over Swift. Swift has a truly

[2] https://www.cappuccino.dev/learn/objective-j.html

[3] /blog/d-or-what-go-may-have-been/

[4] /blog/microservices-or-not-your-team-has-already-decided/

[5] /blog/reusing-apps-between-teams-and-environments-through-containers/

[6] /tags/boring-tech/

[7] https://mjtsai.com/blog/2014/10/14/hypothetical-objective-c-3-0/

[8] https://www.hackingwithswift.com/articles/27/why-many-developers-still-prefer-objective-c-to-swift

> massive hype train, and you definitely don't want to stand in the way of it.

I never really enjoyed writing Swift code. I can't help but to see in it nothing else than the arrogance[9] of a company that used to beg for developers to pay attention to it.

In 2009, the iPhone simulator launched in milliseconds while[10] the Android emulator took ages to do so. In 2017 it was exactly the opposite[11]. Funny how the tables have turned.

Anyway[12], what do I care.

[9] /blog/courage/

[10] /blog/that-mobile-programming-mess/

[11] /books/Android_for_iOS_Devs/Android_for_iOS_Devs_Kotlin_Ed_2018.html#chapter_toolchain

[12] /blog/migrating-from-macos-to-linux/

Stockholm Syndrome in Software

Adrian Kosmaczewski

2022-12-16

Developers working for a particular vendor tend to develop a bizarre version of Stockholm syndrome[1]. It's something I've witnessed at least twice in my career.

- Windows developers who put up with the worst times of MS technology around the end of the 2000s, migrating[2] to Ruby on Rails, the Mac, or wherever possible.
- Apple iOS and macOS developers putting up with buggy, incomplete, undocumented Swift, SwiftUI, and Xcode versions. One day I had enough[3], and some time later I migrated[4] somewhere else.

Those developers will not only put up with the atrocious developer experiences crafted by their alma mater companies; they will even defend those horrors in the name of market reach, security, marketing, design, or any other reason. By "defend," I even mean openly attacking those who dare criticize their technology choices.

I quoted Steve Troughton-Smith last week[5] talking precisely about this issue:

> A little? It's deeply unpopular to use ObjC or say you like ObjC over Swift. Swift has a truly massive hype train, and you definitely don't want to stand in the way of it.

This is precisely the point: choosing a technology for a living is challenging and requires a significant investment from developers: courses, training, books, and lots of practice. All of that takes years to complete, so imagine your reaction when you find out your platform vendor is slowly but surely eroding your capacity to deliver quality software in any way.

So developers clutch at whatever straws they can find to justify their decisions and defend their bread-winning skills. Understandable.

[1] https://en.wikipedia.org/wiki/Stockholm_syndrome
[2] /blog/migration-the-return/
[3] /blog/courage/
[4] /blog/migrating-from-macos-to-linux/
[5] /blog/what-objective-c-3.0-could-have-been/

I preferred to migrate to other galaxies[6], and I have done that at least twice. I will most probably, do that again. I am not attached to a particular tech, even though it might appear like it's the case.

In the case of people working with (or instead suffering because of) Apple crap, I can only feel pain. For example, every time I see on Mastodon[7] good friends dealing with that joke of IDE that is Xcode (in a complete state of destruction since version 4 circa 2010) or the Swift programming language. A language whose versions 1 and 2, let's admit this, were full alpha versions, absolutely not ready for prime time, and probably the crappiest versions of anything ever delivered by Cupertino.

And to add insult to injury, this week JetBrains announced they would stop producing and supporting AppCode[8]. Really bad news; it was a terrific product, unfortunately it never got enough traction, partially thanks to Apple making it impossible for third-parties to provide viable alternatives to Xcode. I really hope Swift Studio[9] by Marcin Krzyżanowski[10] will have better luck.

Speaking about botched developer experiences, Visual Studio .NET ranks high, particularly in its 2 or 3 first releases. It was a clumsy set of tools that could barely remain running for a few minutes before crashing into oblivion. The alternative to .NET was to go back to COM components in various forms, so yes, I'd instead try to get this ASP.NET app to compile and maybe even run; thankyousomuch.

I've seen too many failed projects based on C# 1.1 or Swift 2 for my taste. Both Microsoft and Apple, 13 years apart, played the same game, using their users as testers[11]. It's a shameful situation happening once and again in an industry that likes to make you hate every second of your breathing life as a developer.

No wonder many developers go back to the command line[12], vim, Emacs, and bash; they all certainly propose a limited experience compared to visual tools, but one that at least is stable, predictable, reproducible, and straightforward. Boring[13], even.

I could say the same about FOSS code in general. There's no marketing involved; you know it's going to crash, eventually, anyway. But at least you didn't have to listen to vain promises beforehand. And you will still be able to compile and run your code 20 years from now. Not a tiny proposition, indeed; try to run that 2014 Swift 1 code or that C# 1.0 code from 2001 nowadays; good luck with that.

---

[6] /blog/the-developer-guide-to-migrate-across-galaxies/

[7] /blog/mastodon/

[8] https://blog.jetbrains.com/appcode/2022/12/appcode-2022-3-release-and-end-of-sales-and-support/

[9] https://swiftstudio.app/

[10] https://krzyzanowskim.com/

[11] /blog/users-as-testers/

[12] https://github.com/readme/featured/future-of-the-command-line

[13] /tags/boring-tech/

Stockholm syndrome in software is directly proportional to the hypocrisy and hubris displayed by vendors on their marketing campaigns. At some point, the coin drops, and developers move on somewhere else. It can take a while, depending on the size of your investment in said technology. Sadly, many good developers leave the industry altogether because of these situations, and that's a net loss for all of us. As I said once[14], we need less evangelization, and much more honestization.

[14] https://deprogrammaticaipsum.com/less-evangelization-more-honestization/

Best Books of 2021

Adrian Kosmaczewski

2022-12-23

In 2007[1], 2008[2], 2009[3], 2010[4], 2011[5], 2012[6], and 2013[7] I published lists of books I enjoyed every year. Then I published the list of those I read from 2014 to 2019[8]. I did not keep any records about 2020, that's sad.

Here's the complete list of books I read and enjoyed in 2021. There's a little bit of everything, from poetry to programming to business to philosophy. You can check what I'm reading right now[9] if you're interested, too.

- A Source Book in Mathematics[10], by David Eugene Smith.
- An Introduction to Statistical Learning[11] by Gareth James, Daniela Witten, Trevor Hastie, and Robert Tibshirani.
- Becoming a Technical Leader[12], by Gerald M. Weinberg.
- Business Dynamics[13] by John D. Sterman.
- Cloud Without Compromise[14] by Paul Zikopoulos and Christopher Bienko.
  - Here are some notes I took[15] while reading it.
- Designing Secure Software[16] by Loren Kohnfelder.
- Effective Python[17] by Brett Slatkin.
- El Libro de Arena[18] by Jorge Luis Borges.

[1] /blog/best-books-of-2007
[2] /blog/best-books-of-2008
[3] /blog/best-books-of-2009
[4] /blog/best-books-of-2010
[5] /blog/best-books-of-2011
[6] /blog/best-books-of-2012
[7] /blog/best-books-of-2013
[8] /blog/best-books-of-2014-to-2019
[9] /now/
[10] https://openlibrary.org/books/OL6736284M/A_source_book_in_mathematics
[11] https://web.stanford.edu/~hastie/ISLRv2_website.pdf
[12] https://leanpub.com/becomingatechnicalleader
[13] http://jsterman.scripts.mit.edu/Business_Dynamics.html
[14] https://www.ibm.com/resources/books/cloud-without-compromise/
[15] /blog/notes-about-cloud-without-compromise/
[16] https://designingsecuresoftware.com/
[17] https://effectivepython.com/
[18] https://en.wikipedia.org/wiki/The_Book_of_Sand_(short_story_collection)

- Freakonomics[19] by Steven Levitt and Stephen Dubner.
- In Pursuit of the Unknown: 17 Equations That Changed the World[20] by Ian Stewart.
- Knative in Action[21] by Jacques Chester.
- La Invención de Morel[22] by Adolfo Bioy Casares.
- Principles[23] by Ray Dalio.
- Programmed Inequality[24] by Mar Hicks.
- Speak to Win[25] by Brian Tracy.
- Stancliffe's Hotel[26] by Charlotte Brontë.
- The Goal[27] by Eliyahu M. Goldratt.
- The Problems of Philosophy[28] by Bertrand Russell.
- Thus Spoke Zarathustra[29] by Friedrich Nietzsche.
- Understanding Cryptography[30] by Christof Paar and Jan Pelzl.
- Who Moved My Cheese?[31] by Spencer Johnson.
- Will[32] by Will Smith.

[19] https://en.wikipedia.org/wiki/Freakonomics
[20] https://en.wikipedia.org/wiki/In_Pursuit_of_the_Unknown
[21] https://www.manning.com/books/knative-in-action
[22] https://en.wikipedia.org/wiki/The_Invention_of_Morel
[23] https://www.principles.com/
[24] https://mitpress.mit.edu/books/programmed-inequality
[25] https://www.briantracy.com/catalog/speak-to-win
[26] https://www.penguin.co.uk/books/292/292313/stancliffe-s-hotel/9780241251706.html
[27] https://en.wikipedia.org/wiki/The_Goal_(novel)
[28] https://standardebooks.org/ebooks/bertrand-russell/the-problems-of-philosophy
[29] https://en.wikipedia.org/wiki/Thus_Spoke_Zarathustra
[30] https://www.crypto-textbook.com/
[31] https://en.wikipedia.org/wiki/Who_Moved_My_Cheese%3F
[32] https://willthebook.com/

Shows and Movies of 2022

Adrian Kosmaczewski

2022-12-30

Here is the list of movies and shows I watched (or re-watched) this year. I do not recommend them all; some of it was great, some was utterly forgettable.

First, a small list with what I consider the outstanding, unforgettable, extraordinary, must-watch stuff. In English, Spanish, French, Italian, and German. I seriously recommend these titles.

- 12 Angry Men[1]
- À Bout de Souffle[2]
- Amarcord[3]
- Archive 81[4]
- Being the Ricardos[5]
- Blonde[6]
- Casi Feliz[7]
- Connasse, princesse des coeurs[8]
- Das Leben der Anderen[9]
- Druk[10]
- Granizo[11]
- House of Gucci[12]
- Intimidad[13]
- La Jetée[14]
- Malèna[15]
- Monterey Pop[16]

[1] https://www.imdb.com/title/tt0050083/
[2] https://www.imdb.com/title/tt0053472/
[3] https://www.imdb.com/title/tt0071129/
[4] https://www.imdb.com/title/tt13365348/
[5] https://www.imdb.com/title/tt4995540/
[6] https://www.imdb.com/title/tt1655389/
[7] https://www.imdb.com/title/tt12146940/
[8] https://www.imdb.com/title/tt4466362/
[9] https://www.imdb.com/title/tt0405094/
[10] https://www.imdb.com/title/tt10288566/
[11] https://www.imdb.com/title/tt16427718/
[12] https://www.imdb.com/title/tt11214590/
[13] https://www.imdb.com/title/tt14463542/
[14] https://www.imdb.com/title/tt0056119/
[15] https://www.imdb.com/title/tt0213847/
[16] https://www.imdb.com/title/tt0064689/

- One Day at a Time[17]
- Se7en[18]
- The Batman[19]
- The Piano[20]
- Tina[21]
- Tschugger[22]
- Wolkenbruchs wunderliche Reise in die Arme einer Schickse[23]

The rest, well, it’s up to you. There’s a bit of everything.

- 1899[24]
- A Discovery of Witches[25]
- A Dog’s Journey[26]
- Andor[27]
- Back to the Outback[28]
- Baywatch[29]
- Best Shape of My Life[30]
- Black Panther: Wakanda Forever[31]
- Blasted[32]
- Call Me by Your Name[33]
- Clueless[34]
- Death on the Nile[35]
- Death to 2021[36]
- Destino[37]
- Détox[38]
- Di que sí[39]
- Dirty Grandpa[40]

[17] https://www.imdb.com/title/tt5339440/
[18] https://www.imdb.com/title/tt0114369/
[19] https://www.imdb.com/title/tt1877830/
[20] https://www.imdb.com/title/tt0107822/
[21] https://www.imdb.com/title/tt8399720/
[22] https://www.imdb.com/title/tt15425948/
[23] https://www.imdb.com/title/tt4766630/
[24] https://www.imdb.com/title/tt9319668/
[25] https://www.imdb.com/title/tt2177461/
[26] https://www.imdb.com/title/tt8385474/
[27] https://www.imdb.com/title/tt9253284/
[28] https://www.imdb.com/title/tt13575806/
[29] https://www.imdb.com/title/tt1469304/
[30] https://www.imdb.com/title/tt14583656/
[31] https://www.imdb.com/title/tt9114286/
[32] https://www.imdb.com/title/tt14866710/
[33] https://www.imdb.com/title/tt5726616/
[34] https://www.imdb.com/title/tt0112697/
[35] https://www.imdb.com/title/tt7657566/
[36] https://www.imdb.com/title/tt16301388/
[37] https://www.imdb.com/title/tt0377770/
[38] https://www.imdb.com/title/tt21448608/
[39] https://www.imdb.com/title/tt0377051/
[40] https://www.imdb.com/title/tt1860213/

- Do Revenge[41]
- Doctor Strange in the Multiverse of Madness[42]
- Drôle[43]
- Emily in Paris[44]
- Encanto[45]
- Eternals[46]
- Far from the Tree[47]
- From Scratch[48]
- Grace and Frankie[49]
- Halftime[50]
- Happiest Season[51]
- Harry Potter 20th Anniversary: Return to Hogwarts[52]
- Hocus Pocus 2[53]
- House of the Dragon[54]
- Huit Rue de l'Humanite[55]
- I Am Groot[56]
- I Love America[57]
- Inside the Mind of a Cat[58]
- Johnny Hallyday: Born Rocker[59]
- Le grand bain[60]
- LEGO Star Wars Summer Vacation[61]
- Les goûts et les couleurs[62]
- Lightyear[63]
- Love & Gelato[64]
- Love in the Villa[65]

[41] https://www.imdb.com/title/tt13327038/
[42] https://www.imdb.com/title/tt9419884/
[43] https://www.imdb.com/title/tt18260550/
[44] https://www.imdb.com/title/tt8962124/
[45] https://www.imdb.com/title/tt2953050/
[46] https://www.imdb.com/title/tt9032400/
[47] https://www.imdb.com/title/tt14927356/
[48] https://www.imdb.com/title/tt11252254/
[49] https://www.imdb.com/title/tt3609352/
[50] https://www.imdb.com/title/tt19637852/
[51] https://www.imdb.com/title/tt8522006/
[52] https://www.imdb.com/title/tt16116174/
[53] https://www.imdb.com/title/tt11909878/
[54] https://www.imdb.com/title/tt11198330/
[55] https://www.imdb.com/title/tt13834006/
[56] https://www.imdb.com/title/tt13623148/
[57] https://www.imdb.com/title/tt14867318/
[58] https://www.imdb.com/title/tt21340412/
[59] https://www.imdb.com/title/tt14465966/
[60] https://www.imdb.com/title/tt7476116/
[61] https://www.imdb.com/title/tt20784210/
[62] https://www.imdb.com/title/tt6290418/
[63] https://www.imdb.com/title/tt10298810/
[64] https://www.imdb.com/title/tt15521050/
[65] https://www.imdb.com/title/tt15463032/

- Manifest[66]
- Marie-Francine[67]
- Moon Knight[68]
- Ms. Marvel[69]
- Napo[70]
- Nine Lives[71]
- No Time to Die[72]
- Obi-Wan Kenobi[73]
- Palais royal![74]
- Pam & Tommy[75]
- Persuasion[76]
- Ready Player One[77]
- Rough Night[78]
- Second Act[79]
- Senior Year[80]
- She-Hulk: Attorney at Law[81]
- Stranger Things[82]
- Superlópez[83]
- That's Amor[84]
- The Adam Project[85]
- The Bastard Son & The Devil Himself[86]
- The Book of Boba Fett[87]
- The Boys[88]
- The Crown[89]
- The Curse of Bridge Hollow[90]

[66] https://www.imdb.com/title/tt8421350/
[67] https://www.imdb.com/title/tt6081632/
[68] https://www.imdb.com/title/tt10234724/
[69] https://www.imdb.com/title/tt10857164/
[70] https://www.imdb.com/title/tt11918504/
[71] https://www.imdb.com/title/tt4383594/
[72] https://www.imdb.com/title/tt2382320/
[73] https://www.imdb.com/title/tt8466564/
[74] https://www.imdb.com/title/tt0424338/
[75] https://www.imdb.com/title/tt13659418/
[76] https://www.imdb.com/title/tt13456318/
[77] https://www.imdb.com/title/tt1677720/
[78] https://www.imdb.com/title/tt4799050/
[79] https://www.imdb.com/title/tt2126357/
[80] https://www.imdb.com/title/tt5315212/
[81] https://www.imdb.com/title/tt10857160/
[82] https://www.imdb.com/title/tt4574334/
[83] https://www.imdb.com/title/tt4110388/
[84] https://www.imdb.com/title/tt21388556/
[85] https://www.imdb.com/title/tt2463208/
[86] https://www.imdb.com/title/tt13649396/
[87] https://www.imdb.com/title/tt13668894/
[88] https://www.imdb.com/title/tt1190634/
[89] https://www.imdb.com/title/tt4786824/
[90] https://www.imdb.com/title/tt15289240/

- The French Dispatch[91]
- The Greatest Showman[92]
- The Hidden Lives of Pets[93]
- The Kardashians[94]
- The Lord of the Rings: The Rings of Power[95]
- The Man from Toronto[96]
- The Sandman[97]
- The School for Good and Evil[98]
- The Secret Life of Pets 2[99]
- Thor: Love and Thunder[100]
- Toscana[101]
- Uncoupled[102]
- Under the Helmet: The Legacy of Boba Fett[103]
- Vacation[104]
- Vivir dos veces[105]
- What to Expect When You're Expecting[106]
- White Hot: The Rise & Fall of Abercrombie & Fitch[107]

[91] https://www.imdb.com/title/tt8847712/
[92] https://www.imdb.com/title/tt1485796/
[93] https://www.imdb.com/title/tt20560384/
[94] https://www.imdb.com/title/tt15791630/
[95] https://www.imdb.com/title/tt7631058/
[96] https://www.imdb.com/title/tt11671006/
[97] https://www.imdb.com/title/tt1751634/
[98] https://www.imdb.com/title/tt2935622/
[99] https://www.imdb.com/title/tt5113040/
[100] https://www.imdb.com/title/tt10648342/
[101] https://www.imdb.com/title/tt13276352/
[102] https://www.imdb.com/title/tt15466144/
[103] https://www.imdb.com/title/tt15715890/
[104] https://www.imdb.com/title/tt1524930/
[105] https://www.imdb.com/title/tt9063902/
[106] https://www.imdb.com/title/tt1586265/
[107] https://www.imdb.com/title/tt19034522/

Best Books of 2022

Adrian Kosmaczewski

2023-01-06

In 2007[1], 2008[2], 2009[3], 2010[4], 2011[5], 2012[6], and 2013[7] I published lists of books I enjoyed every year. Then I enumerated those I read from 2014 to 2019[8], and then the ones of 2021[9].

Here's the complete list of books I read and enjoyed in 2022. There's a little bit of everything, from poetry to programming to business to philosophy. You can check what I'm reading right now[10] if you're interested, too.

- Beginning COBOL for Programmers[11] by Michael Coughlan.
- Data and Reality[12] by William Kent.
- HBR at 100: The Most Influential and Innovative Articles from Harvard Business Review's First Century[13] by Harvard Business Review.
- Home Computers: 100 Icons that Defined a Digital Generation[14] by Alex Wiltshire.
- IBM: The Rise and Fall and Reinvention of a Global Icon[15] by James W. Cortada.
- Introduction to Compilers and Language Design[16] by Douglas Thain.
- Modern Mainframe Development[17] by Tom Taulli.

[1] /blog/best-books-of-2007
[2] /blog/best-books-of-2008
[3] /blog/best-books-of-2009
[4] /blog/best-books-of-2010
[5] /blog/best-books-of-2011
[6] /blog/best-books-of-2012
[7] /blog/best-books-of-2013
[8] /blog/best-books-of-2014-to-2019
[9] /blog/best-books-of-2021
[10] /now/
[11] https://www.goodreads.com/book/show/21539568-beginning-cobol-for-programmers
[12] https://cmpct.info/~calvin/Papers/Data%20and%20Reality.pdf
[13] https://store.hbr.org/product/hbr-at-100-the-most-influential-and-innovative-articles-from-harvard-business-review-s-first-century/10557
[14] https://mitpress.mit.edu/9780262044011/home-computers/
[15] https://mitpress.mit.edu/9780262039444/ibm/
[16] https://www3.nd.edu/~dthain/compilerbook/
[17] https://www.oreilly.com/library/view/modern-mainframe-development/97810981

- Obras Completas II[18] by Jorge Luis Borges.
- On Writing: A Memoir of the Craft[19] by Stephen King.
- Otras Inquisiciones[20] by Jorge Luis Borges.
- Principles for Dealing with the Changing World Order[21] by Ray Dalio.
- Recoding Gender[22] by Janet Abbate.
- The Art of Literature[23] by Arthur Schopenhauer.
- The Business Bottleneck[24] by Michael Coté.
- The Nurnberg Funnel: Designing Minimalist Instruction for Practical Computer Skill[25] by John M. Carroll.
- Think Again[26] by Adam Grant.
- Wonderland[27] by Annie Leibovitz.

07017/

[18] https://en.wikipedia.org/wiki/Jorge_Luis_Borges

[19] https://stephenking.com/works/nonfiction/on-writing-a-memoir-of-the-craft.html

[20] https://es.wikipedia.org/wiki/Otras_inquisiciones

[21] https://www.principles.com/the-changing-world-order/

[22] https://mitpress.mit.edu/books/recoding-gender

[23] https://www.gutenberg.org/ebooks/10714

[24] https://tanzu.vmware.com/content/ebooks/the-business-bottleneck

[25] https://mitpress.mit.edu/books/nurnberg-funnel

[26] https://www.adamgrant.net/book/think-again/

[27] https://www.amazon.com/Annie-Leibovitz-Wonderland/dp/1838661522

Firefox Extensions

Adrian Kosmaczewski

2023-01-13

Everybody uses Chrome, but I don't; Google is too powerful already, no need to feed the beast anymore. At most, I have a copy of Chromium[1] installed, as a last resort. Because I'm a faithful Firefox[2] user, and of course I install many extensions in it. Here are some of those that I systematically add to every computer I use.

- Dark Reader[3] so that web pages automatically adapt to the current "Night Mode" of the computer. The official website[4] explains it all, it's a great add-on.
- Facebook Container[5] because I really don't want Facebook to track me at all.
- GNOME Shell Integration[6] is a great way to easily install GNOME[7] extensions in my system. But for this to work I remove the snap installation of Firefox, and install it via `apt`. I actually remove snap altogether from Ubuntu, if you ask me.
- Grammarly[8] to make sure that what I write on the web is decent enough (unfortunately it's the only piece of the Grammarly[9] system that actually works on Linux.)
- Joplin Web Clipper[10] to save web pages into a Joplin[11] notebook.
- KeePassXC Browser[12] providing me with access to my KeePassXC[13] database. Again, for this to work properly, both Firefox and KeePassXC need to be installed via `apt`, and not via `snap` of Flatpak[14].
- uBlock Origin[15] because it makes the web lighter and faster.

[1] https://www.chromium.org/
[2] https://www.mozilla.org/en-US/firefox/
[3] https://addons.mozilla.org/en-US/firefox/addon/darkreader/
[4] https://darkreader.org/
[5] https://addons.mozilla.org/en-US/firefox/addon/facebook-container/
[6] https://addons.mozilla.org/en-US/firefox/addon/gnome-shell-integration/
[7] https://www.gnome.org/
[8] https://addons.mozilla.org/en-US/firefox/addon/grammarly-1/
[9] https://www.grammarly.com/
[10] https://addons.mozilla.org/en-US/firefox/addon/joplin-web-clipper/
[11] https://joplinapp.org/
[12] https://addons.mozilla.org/en-US/firefox/addon/keepassxc-browser/
[13] https://keepassxc.org/
[14] various-flatpak-tips-and-tricks
[15] https://addons.mozilla.org/en-US/firefox/addon/ublock-origin/

Also, I don’t know if many people know about this, but there’s a “Chromium equivalent” for Firefox, called LibreWolf[16], and you might want to check it out.

[16] https://librewolf.net/

GNOME Extensions

Adrian Kosmaczewski

2023-01-20

Last week I talked about Firefox extensions[1], last year I talked about VSCode extensions[2], now it's time to talk about GNOME extensions. For those who don't know, GNOME[3] is the name of one of the various desktop environments available for Linux and other operating systems.

Here are a few extensions I systematically install in every computer I use.

- Night Theme Switcher[4]
- GNOME Shell Frippery[5] to move the clock to the right side of the menu bar, instead of it sitting in the center.
- Audio Output Switcher[6] to be able to switch audio output easily.
- Open Weather[7]
- Applications Menu[8]
- Emoji Selector[9]

I also used NoAnnoyance[10] in the past, to avoid the "window is ready" popup and to have windows coming to the foreground when opened… but it does not work in Ubuntu 22.04.

If you are a Firefox + GNOME user, the GNOME Shell Integration[11] extension for Firefox is a great way to easily install GNOME extensions in your system. But remember what I said last week[12]:

> But for this to work I remove the snap installation of Firefox, and install it via `apt`. I actually remove snap altogether from Ubuntu, if you ask me.

[1] /blog/firefox-extensions/
[2] /blog/lots-of-vscode-extensions/
[3] https://www.gnome.org/
[4] https://nightthemeswitcher.romainvigier.fr/
[5] https://frippery.org/extensions/
[6] https://github.com/adaxi/audio-output-switcher
[7] https://extensions.gnome.org/extension/750/openweather/
[8] https://extensions.gnome.org/extension/6/applications-menu/
[9] https://extensions.gnome.org/extension/1162/emoji-selector/
[10] https://extensions.gnome.org/extension/1236/noannoyance/
[11] https://addons.mozilla.org/en-US/firefox/addon/gnome-shell-integration/
[12] /blog/firefox-extensions/

Update, 2023-05-19: As I've moved to Fedora 38 and GNOME 44, I'm using Quick Settings Audio Panel[13] instead of the Audio Output Switcher, and AppIndicator and KStatusNotifierItem Support[14] to get tray icons back.

[13] https://extensions.gnome.org/extension/5940/quick-settings-audio-panel/
[14] https://extensions.gnome.org/extension/615/appindicator-support/

FOSS in Developing Countries

Adrian Kosmaczewski

2023-01-27

The other day I had friends in Bolivia asking me if they could install Windows on a laptop they got through an NGO that initially came bundled with Linux.

I know people in Argentina who buy Macs and install pirated copies of Photoshop and Adobe Premiere and would not even consider using anything else.

These days I hear Twitter users complaining that Mastodon offers different features than Twitter and thus is not even worth considering.

I often hear non-geeks saying that Linux and LibreOffice are no good for them because jobs require people to know Windows and MS Office. They won't even try them.

In my experience, people would rather keep using a virus-laden pirated copy of Windows than learn how to use GNOME, GIMP[1], Inkscape[2], KDEnlive[3], LibreOffice[4], or whatever free software you could `apt get` in your machine.

My take on this situation is a highly unpopular one. Here it goes. You've been warned.

The FOSS movement is a "First World" thing. Its appeal is limited to the small number of elites working in the computer industry in developed countries and ironically inaccessible to the population of those countries who paradoxically would need it the most.

There, I said it.

Let me rephrase: Linux and the whole FOSS movement are just luxury items directed to a small fraction of the human population, despite their seemingly universal appeal.

It's like being vegan; you will find few in countries where food is scarce. Go to a wealthy city in a developing country, and you will find plenty of vegan shops and restaurants. To a large degree, using FOSS is like

[1] https://www.gimp.org/
[2] https://inkscape.org/
[3] https://kdenlive.org/en/
[4] https://www.libreoffice.org/

being vegan, and both groups are equally vocal about their ethical choices.

Being vegan is a privilege. Being able to use Linux is a privilege.

Let's be honest. To run Linux properly, you either need to know quite a bit (a lot, even) about computers or buy a laptop from a well-known brand like HP, Dell, or Lenovo. Try installing Linux in a PC built with components sourced in some remote end of the world: good luck. You may be lucky and find drivers for those. Maybe not.

Macs, Lenovo, and HP laptops are more expensive, but you get machines that work. You can't get that with Linux; it's a gamble at best, even if it's much better than 20 years ago. Nowadays, you can even find vendors selling laptops with pre-installed Linux[5] in Europe and the USA, but I need to find something similar in Latin America. Maybe there are. I hope there are.

Don't get me wrong; Linux is a godsend in many other contexts. Just not in this particular one.

Let's go beyond the "can people install Linux on their laptop" question. Can people get a job with Linux or LibreOffice in their resumé? Nope–well, unless they apply for an IT job, of course. Can they get one if they mention Windows or MS Office? You bet. It's something that people in the developed world (USA, Japan, Western Europe) will never truly understand.

Geeks are, of course, thrilled to use Linux anywhere in the world. There's a thriving community of Linux users all across Latin America. But by and large, the (admittedly more significant) non-geek part of the population would instead install pirated commercial software and run it on a virus-laden laptop with an illegal copy of Windows than take the time to move to Linux. It's a fact.

Even with Wine[6] or Darling[7], running Windows (respectively, Mac) software on Linux boxes is almost impossible. Why use Linux when you can just run Windows or macOS directly for free? In concrete terms, for most of the planet, in those contexts, not much.

Linux and its whole world of associated software is just a first-world problem or, at most, a luxury, a privilege only affordable to IT experts.

[5] /blog/computers-bundled-with-linux/
[6] https://www.winehq.org/
[7] https://www.darlinghq.org/

Javascript Animations

Adrian Kosmaczewski

2023-02-03

Around 1998, Macromedia Dreamweaver[1] allowed developers to create animated web pages using 100% generated JavaScript code. This was before we were told about the good parts[2] of JavaScript, before script.aculo.us[3], and without the need for Flash players[4] or Java applets[5] of any kind.

Dreamweaver used a similar approach to Flash[6], in the sense that it offered a non-linear timeline, where you could define keyframes with specific properties (colors, positions on the screen, etc.) and Dreamweaver would generate the required JavaScript code (compatible with both Netscape and Internet Explorer) to create said animation.

The final animation was very, very, very similar to those generated by the above mentioned script.aculo.us[7], but sadly (and understandably, for commercial reasons) not reusable outside of Dreamweaver. This means... well, good luck understanding the generated JavaScript code. It was quite obfuscated (think webpack[8] or similar) even if not completely minified, but in any case one could see a big `if` statement at the beginning:

```
if (IE) {
    // Internet Explorer code
}
else if (Netscape) {
    // Netscape code
}
```

Every page generated by Dreamweaver would contain its own generated code, which invariably represented a large amount of JavaScript to be downloaded inside of each page. Well, large, compared to the

[1] https://en.wikipedia.org/wiki/Adobe_Dreamweaver

[2] https://deprogrammaticaipsum.com/douglas-crockford/

[3] https://script.aculo.us/

[4] https://en.wikipedia.org/wiki/Adobe_Flash_Player

[5] https://en.wikipedia.org/wiki/Java_applet

[6] Which makes sense, because both came from the same company, Macromedia, bought by Adobe in 2005.

[7] https://script.aculo.us/

[8] https://webpack.js.org/

standards of that era, and to the relative speeds of 28k and 56k modems[9] of the day. These days we wouldn't even notice them. Think about the regular amounts of React and Angular code we have to download every day…

And the animations were simply flawless. A very clever use of `setTimeout()` and `setInterval()` to handle animations without blocking the single-threaded nature of JavaScript execution.

All of this happened seven years before Prototype[10], eight years before jQuery[11]. Both projects demonstrated that encapsulating the `if` statement inside of a library and providing a common interface was the way to provide a true cross-platform experience to web developers. The rest is history.

Dreamweaver was in itself a joy of an editor, with unparalleled features (particularly when compared with that clunky competitor called FrontPage[12]), but this animation editor certainly was the cherry on top of a fantastic cake.

[9] https://en.wikipedia.org/wiki/Modem
[10] https://en.wikipedia.org/wiki/Prototype_JavaScript_Framework
[11] https://en.wikipedia.org/wiki/JQuery
[12] https://en.wikipedia.org/wiki/Microsoft_FrontPage

Server Side Rendering FTW

Adrian Kosmaczewski

2023-02-10

I am, I have been, and forever will be a big advocate of server-side rendering. I think it is an essential way to build dynamic web content. I believe in this adamantly, feverishly, strongly, and relentlessly.

JavaScript is not made for rendering HTML. The fact that it can modify the DOM does not mean you have to use it for that. Just don't. HTTP requests are meant to return HTML. Web browsers render HTML just fine. Do that.

And I'm not the only one[1] who thinks like that. We have been lied by a full industry claiming that using JavaScript on the client (and the server! FFS!) to request JSON data, and then to manipulate the DOM accordingly, was the best possible way to create web applications.

What. The. Actual. Fuck.

The creators of those "SPA frameworks" actually bullied people[2] who dared speak the truth of the utter insanity of using such systems. Talk about mafia tactics.

The results of SPAs are abysmal. The web has crawled to a halt, crippling the user experience of everything we use on the web.

The drawbacks for end-users are many:

- Broken back button functionality.
- Nonworking stop buttons.
- Really slow updates when state change.
- Inconsistent page states.
- Bookmarking and sharing URLs with others.

And to add insult to injury, a total lack of proper error handling and feedback when network connections die. Which hey! Happens more often that we would like to acknowledge in 2023.

Even smaller apps, that would be best served with server-side rendering, fall into the trap of frontend evangelists trying to sell their shit.

Ironically, the "IT Crowd"-like solution for this kafkaesque situation consists of… reloadinng the whole page once again. So much for a

[1] https://infrequently.org/2023/02/the-market-for-lemons/
[2] https://fediverse.zachleat.com/@zachleat/109830047951867907

“single page application,” dammit. Remember when your navigator would do that for you every time you navigated from one page to the other? Yeah.

And even for developers, the tooling is maddeningly slow. JavaScript is a bad choice for this. I can rebuild this website from scratch using Hugo[3] in less than a second. On the other hand, frontend applications take ages to build, using half-assed tools like gulp, npm, or whatever the build tool du jour.

Oh! But you can share (usually a few small, quite insignificant data classes) JavaScript between both client and server. Wohooooooo!

We’re a brain-dead industry. I’m fucking tired.

Making web apps? Mustache templates[4] and similar technologies are the way to go. They have to reclaim their due place as the proper mechanism to render HTML on the server. Using programming languages like Go, Crystal, C#, or Rust, and so many others[5], means you can have blazingly fast performance on the server, while letting browsers do what they do best: display HTML and CSS.

And yes, if you must, sprinkle some JavaScript here and there if it really helps. Not just for the sake of it; but when it adds real value to the end user experience.

For how long are we going to have to cope with the shitty web experience of 2023?

Update, 2023-02-17: More[6] about the subject of stopping with this SPA madness.

[3] https://gohugo.io/
[4] https://mustache.github.io/
[5] /blog/fortune-apps/
[6] https://dev.to/oxharris/rethinking-the-modern-web-5cn1

Those Function Keys

Adrian Kosmaczewski

2023-02-17

There is a somewhat unspoken standard in the functionality exposed by function keys in Windows and Linux, only partially shared with macOS, particularly since the introduction of that abomination called the Touch Bar. Here is a very simple recap of those functions.

- F1 = Get help.
- F2 = Edit or rename the current selection.
- F3 = Open the search dialog box, or repeat the previous action.
- F4 = Open the find dialog box.
- F5 = Refresh the screen, or recalculate.
- F6 = Select the browser bar in most browsers.
- F7 = Trigger spell and grammar check in some text editors.
- F10 = Open the menu bar in various apps, like Emacs and Firefox.
- F11 = Toggle fullscreen mode in many apps in Windows and Linux.
- F12 = Open the “save as” dialog.

In developer-centric apps, the keys from F5 to F8 also serve to launch and debug applications, moving from one instruction to the next and eventually jumping in and out of functions.

Update, 2023-12-29: Maybe what I’m looking for is CUA[1]?

[1] https://en.wikipedia.org/wiki/IBM_Common_User_Access

Yup, Still Learning a New Programming Language Every Year

Adrian Kosmaczewski

2023-02-24

I gave an update on this lifetime activity of mine in 2006[1], 2007[2], 2011[3], and 2013[4], and here we go for 2023.

The languages featured in my Conway project[5] appear in bold.

- 1992: **QBasic**
- 1993: **Turbo Pascal**
- 1994: **C**
- 1995: Delphi
- 1996: **JavaScript**. Others: HTML
- 1997: **Java**
- 1998: VBScript. Others: CSS
- 1999: Transact-SQL
- 2000: **C#**. Others: Prolog
- 2001: **C++**
- 2002: **PHP**. Others: **C#**
- 2003: **Objective-C**
- 2004: **Visual Basic.NET**
- 2005: **Ruby**
- 2006: LINQ. Others: **C# 2.0**, **C++**
- 2007: Erlang
- 2008: **Python**
- 2009: **Go**
- 2010: **Common Lisp**
- 2011: Haskell
- 2012: **Lua**
- 2013: **C++ 11**
- 2014: Scala
- 2015: **Swift**. Others: **C++ 14**
- 2016: **Kotlin**. Others: **Swift 3**, **Java 1.7**
- 2017: **TypeScript**. Others: 68K Assembler, **PHP 7**, awk
- 2018: **F#**. Others: **C# 7**, Elisp

[1] /blog/a-new-programming-language-every-year/
[2] /blog/erlang/
[3] /blog/learning-one-new-language-every-year/
[4] /blog/still-learning-one-language-per-year/
[5] /blog/polyglot-conway/

- 2019: **Go**. Others: Scheme, Ballerina
- 2020: **Smalltalk**. Others: **Go**, Rexx
- 2021: **Rust**. Others: R
- 2022: **Dart**. Others: Elixir, **Crystal**, **Perl**, **D**

Nothing decided for 2023 yet, probably Zig[6].

[6] https://ziglang.org/

Best Review

Adrian Kosmaczewski

2023-03-03

Here's an infuriating short story. One of my books has a one-star review on Amazon, and it literally starts with the phrase "I have not read the book".

I won't link to it, you can find it very easily.

That's it. It's puzzling until you click on the reviewer profile and you realize that most of the reviews this person has written are more or less of the same kind.

If I'm writing this after a decade, it clearly means that I'm still mesmerized by these things.

That's it.

Sizing Exercises Correctly

Adrian Kosmaczewski

2023-03-10

I have often learned technical subjects online or in person with oversized exercises. By that, I mean sample code or applications that are needlessly complex and contrived, to the point that their complexity hides the main subject of the class. Such examples are hard to install, run, and understand, and teachers need to spend more time helping their students to run the code than actually explaining the subject.

By making this mistake, online teachers destroy the purpose of their learning proposition, their main point buried beneath lots of unnecessary code.

What is the right size of a code example? The answer to this question depends on various factors. I do not have a rule of thumb other than the following: make it as small as possible and then some more.

Code samples for online learning must fulfill the following characteristics:

Simple to install. Learners must be able to install and run the code in seconds unless the course is about the installation process itself. If the examples are complex to set up, provide an installer, a "curl-pipe-bash" script, a Helm chart, or any other mechanism that makes deploying code as fast as possible. Your students are here to learn the gist of the code, not to spend the first ten minutes of a 45-minute session setting up their environment. Unless, of course, you're teaching "Kubernetes the Hard Way" or some similar subject.

Short. One hundred lines of code are too much. 50 is too much. Five is not enough. Ten to twenty is about right. Remove any boilerplate code required for the code to run and place it in a different file, on a separate module, or in a library. Make those available to students, but keep the example itself neat. Unless, of course, you're teaching software architecture or some bigger-than-life subject where size is the main topic.

Relevant. This is largely a consequence of the previous point. Do not mix and match clever tips and tricks around the API call or design pattern you want to show. Concentrate on the thing you want to teach; ask yourself, what message conveys this code? What problem is it solving? If the answer is longer than expected, break the code sample into smaller pieces.

No eye-candy. Many teachers spend a lot of time “pinping up” their code examples with eye-catching features, when sometimes just a white screen with some text is enough to get the point across. The visuals on your demos and sample code do not need to win an award. Unless, of course, you’re teaching accessibility, design, or some other related subject.

20 Years of Harman Kardon SoundSticks

Adrian Kosmaczewski

2023-03-17

Twenty years ago, months before the scorching (and deadly) summer of 2003, I bought the transparent Harman Kardon SoundSticks[1] that I still have above (and below) my desk today.

They are the first iteration of the SoundSticks, those introduced at the July 2000 MacWorld expo, and designed by Jony Ive himself. And they still sound amazing today, even after 20 years of continuous use.

They also look fabulous; the two transparents speakers on my desk, at each side of my computer screen, and the subwoofer next to my feet, providing the bass. A humble USB-1 connection brings music to my office.

I have connected them to various computers; starting with my iBook G3 (bought in December 2002), various Macs (with PowerPC, Intel, and ARM chips), all the way to the Windows 11 PC of my wife, as well as my work computers, all running Linux (Ubuntu and RHEL).

I want to point out this very simple fact: no matter which computer I plug them to, they are immediately recognized and they just work without any further configuration. They even appear correctly labeled as “Harman Kardon SoundSticks” in every one of those operating systems.

This is, simply put, how all quality technology should work. Most amazing still, Harman Kardon still sells them[2]. And Forbes magazine even published an article[3] about them.

In these days where we try to consume less in a world filled with planned obsolescence, it’s soothing to find at least one piece of technology still compatible with today’s computers… 20 years later.

I’ve been celebrating the 20 years of my classic SoundSticks listening to a 50 year old classic: Pink Floyd’s The Dark Side of the Moon[4] over and over again. Good things are timeless.

[1] https://en.wikipedia.org/wiki/SoundSticks

[2] https://www.harmankardon.com/home-audio/HK+SOUNDSTICKS+4.html

[3] https://www.forbes.com/sites/marksparrow/2019/04/09/harman-kardon-soundsticks-are-still-the-ideal-apple-mac-partner-some-two-decades-on/?sh=6213ce691f3d

[4] https://en.wikipedia.org/wiki/The_Dark_Side_of_the_Moon

EditPlus

Adrian Kosmaczewski

2023-03-24

Last week I was celebrating[1] the 20 years of my Harman Kardon SoundSticks, but last Monday there was another anniversary that some of us celebrated fondly: EditPlus 1.0[2], released March 20th, 1998, is 25 years old!

EditPlus, made by a programmer from South Korea, was the first editor where I wrote code for a living. In those days my job consisted of writing Active Server Pages[3] with VBScript[4] code. If you don't know what VBScript or ASP look like, think PHP but with an even shittier programming language.

EditPlus was the best editor for ASP + VBScript, hands down. There was no other editor providing a better experience. And, yes, this was 1998, so a "good experience" was merely just good syntax highlighting. There was no debugger, there were no unit tests, it was a mess.

(A few years later somebody came up with ASPUnit[5], but the writing was on the wall: VBScript and ASP will be no more, with C# and ASP.NET replacing them.)

EditPlus provided very good syntax highlighting for those ASP pages, mixing HTML, CSS, client-side JavaScript, and server-side VBScript in the same file. It also had an integrated FTP client, which meant that one could edit files on our server (physically located in a datacenter somewhere in upstate New York) without needing a separate app (my FTP client of choice was WS_FTP LE[6]).

To top it off, EditPlus used the Internet Explorer COM component and offered integrated previews of HTML code directly in the editor. Talk about features.

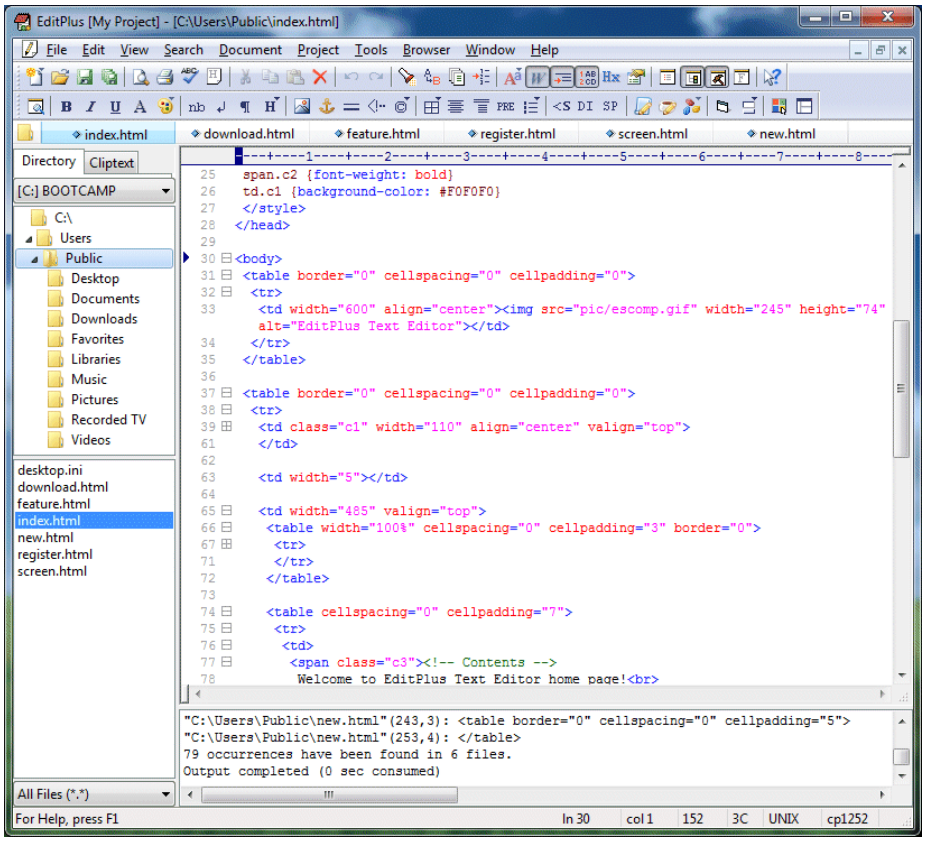
Over the years EditPlus gained lots of extensions[7], named "user files" by the creator of the application. I'm glad to see that more and more of those appeared, even last year! Suprisingly, though, there isn't

[1] /blog/20-years-of-harman-kardon-soundsticks/
[2] https://editplus.com/
[3] https://en.wikipedia.org/wiki/Active_Server_Pages
[4] https://en.wikipedia.org/wiki/VBScript
[5] https://aspunit.sourceforge.net/
[6] https://winworldpc.com/product/ws-ftp/le
[7] https://editplus.com/files.html

one for Rust, at least not yet. There is now support for Ruby on Rails, C#, Dark Mode, Emmet, Markdown, and so many other things that weren't there in 1998.

As I said ASP and VBScript were a mess, but the combo worked; we were serving thousands of simultaneous users on a single CPU Pentium II server running Windows NT 4.0. And yes, the same server was running both SQL Server and the web server (IIS).

So here's me wishing EditPlus, whose latest version[8] 5.7 was released last January in 32 and 64 bits for both Intel and ARM Windows (!) a very happy birthday! Check out the list of features[9], watch some screenshots[10], and follow EditPlus on Mastodon[11].



[8] https://editplus.com/download.html
[9] https://editplus.com/feature.html
[10] https://editplus.com/screen.html
[11] https://twingyeo.kr/@EditPlus

Bootstrap

Adrian Kosmaczewski

2023-03-31

I love Bootstrap[1]. No matter which web frontend framework I try, I always end up returning to it.

There's a lot of things I like about it:

- **Safe defaults;** including a nice grid where you can lay your content and be sure it looks good in every device.
- **Mobile-friendly and cross-browser;** your app will look exactly the same no matter where you look at it.
- **Easily themable;** just customize a few SCSS variables here and there, rebuild the CSS, and you can have dramatically different look & feels.
- **Only use what you need;** just include the CSS if nothing else, or also add the JavaScript if you need some special behaviour.
- **Extensive library of built-in components;** whatever you need to draw on your browser, you just have it right there. Write the HTML, apply the CSS, eventually some JavaScript if really needed, and you're good to go.
- **No magic;** to call this a framework is a stretch, because it technically isn't one. It is a set of CSS and JavaScript that works with a particular set of HTML tags, nothing else. New team members can immediately pick up what's going on (provided they have some knowledge about the saint trinity of web development, HTML, CSS, and JS.)
- **Particularly well-suited for server-side rendering;** which in my opinion is how complex web apps should work. Leave the browser to render HTML and manage history, no need to use JavaScript for that.
- **Great documentation;** not perfect, but it's damn close to perfect. It's well organized and easy to follow. The examples are relevant and it's very easy to have a Bootstrap project up and running in almost no time.

Bootstrap is boring technology[2] and I like that.

[1] https://getbootstrap.com/
[2] /tags/boring-tech/

Redmine

Adrian Kosmaczewski

2023-04-07

I was surprised to discover recently that good old Redmine[1] not only still exists in 2022, but it thrives in various unexpected ways. Redmine is another one of those boring tech things[2] that I love.

When I was running akosma software[3], more than a decade ago, I used to run two instances of it, one for my projects and another for my trainings. It's simple, it works, it does what it has to do, and more. It has an impressive set of features and can be extended in oh so many ways:

- Bug tracker
- Custom enumerations and fields
- Gantt charts
- Calendars
- Users (including LDAP integration)
- Wikis (including attachments of any kind)
- File sharing
- Email notifications
- Source control integration (Git, Subversion…)
- Releases
- Creating issues directly from emails
- …

First Surprise

But we're in 2023, with Kubernetes and Cloud Native apps and 12 Factor[4] apps and whatnot. Let us begin by the obvious first surprise:

```
$ docker run --rm --publish 8080:3000 redmine:5.0.5-alpine
```

Yes, the container works as-is perfectly well, and localhost:8080[5] shows a fully working demo version, accessible with the username and password `admin` which must be changed at first login. This

[1] https://redmine.org/
[2] /tags/boring-tech/
[3] /blog/running-akosma-software/
[4] https://12factor.net/
[5] http://localhost:8080

demo version uses a SQLite database that is not preserved when the container dies.

The container images are stored in Docker Hub[6] with a matching repository full of `Dockerfile` on GitHub[7].

Second Surprise

And the second surprise is that this very container works in OpenShift off-the-box. Not all monolithic apps work so easily on OpenShift, a platform that is quite picky in terms of security settings, and this is a very nice surprise.

To give an example, a few days ago I tried to run the Shlink[8] URL shortener application written in PHP on OpenShift, and to make a long story short: it's impossible. Don't even try. Some apps are just not made to be run on Kubernetes, or not even on containers[9].

In any case, I would totally reuse Redmine again in the future, should I start a small software consultancy business again. And even more now that I see that I could run it in OpenShift.

[6] https://hub.docker.com/_/redmine/

[7] https://github.com/docker-library/redmine

[8] https://shlink.io

[9] The biggest problem with Shlink, in particular, was its (ab)use of configuration files accessed in write mode when the application starts in various locations. Oh, and insisting in using its own PHP-based CLI to initialize API keys and other things. There's a reason why the 12 Factor App website recommends using environment variables instead.

Matomo

Adrian Kosmaczewski

2023-04-14

In the past few weeks I've been making quite a few changes to this website. Some are visible, some less. Among the visible ones, I removed the downloadable PDF files feature, which were taking a lot of space and weren't really that useful. Maybe some of you noticed[1].

But the most important things are hidden from view. First of all, this website is now built and published using a GitLab CI/CD pipeline. That's right, I'm going full DevOps but without Kubernetes. I'll talk more about this in another article.

The other important change is that I have installed an instance of Matomo[3] on this server, "the Google Analytics alternative that protects your data." I have configured this Matomo instance to work in "cookieless" mode[4].

Matomo is extremely powerful, and superbly easy to configure.

1. Copy the files on your server; it's built with PHP, so Filezilla[5] or `scp` are your friends. In my case I just `ssh`'d into my server and then `wget https://builds.matomo.org/matomo.zip && unzip matomo.zip` from there. Much simpler.
2. Create a MySQL database; nothing that Hostpoint[6] or any other decent web hosting couldn't handle.
3. Launch the installation of Matomo; literally, visit `index.php` and complete the wizard.
4. Set a helpful `cron` job on the server to generate reports every so often.
5. Configure the instance to go cookieless in the Administration settings.
6. Include the JavaScript tracker code in the websites you would like to track:

[1] To be honest, it was more an experiment on making a proper `Makefile` driving Pandoc[2] than anything else. Generating the 600+ PDF files for this whole website took about half an hour in my laptop. But the end result was quite nice, with Pandoc even respecting the syntax-highlighting of the code snippets, the images, showing the links as a footer, and much more. Material for another article.

[3] https://matomo.org/

[4] https://matomo.org/faq/how-to/how-do-i-enforce-tracking-without-cookies/

[5] https://filezilla-project.org/

[6] https://www.hostpoint.ch/

```
<!-- Matomo -->
<script>
  var _paq = window._paq = window._paq || [];
  /* tracker methods like "setCustomDimension" should be called before "trackPa
  _paq.push(['trackPageView']);
  _paq.push(['enableLinkTracking']);
  (function() {
    var u="//example.com/matomo/";
    _paq.push(['setTrackerUrl', u+'matomo.php']);
    _paq.push(['setSiteId', '146']);
    var d=document, g=d.createElement('script'), s=d.getElementsByTagName('scri
    g.async=true; g.src=u+'matomo.js'; s.parentNode.insertBefore(g,s);
  })();
</script>
<!-- End Matomo Code -->
```

7. Download and configure the Matomo mobile app[7].

Going cookieless is very important to me, because it means that I keep my promise of not using cookies[8] in this website, while at the same time I can have some information about who's using it and from where in the world. The data I gather is 100% private, available only to me, and I don't sell it to anyone.

So now you know. I'm just curious about where you come from and what devices you're using; but I don't care who you are specifically. And as always, thanks for visiting!

[7] https://matomo.org/mobile/
[8] /blog/no-cookie-popup/

Hugo in DevOps Mode

Adrian Kosmaczewski

2023-04-21

As I explained last week[1] I have been updating this website in various ways; I removed the downloadable PDFs, then added privacy-friendly analytics, and finally, I set up a scheduled pipeline in GitLab to automatically build and deploy this website every Friday morning.

Yes, I admit that for the last 2.5 years[2] I've built and deployed this website manually; the thing is that for a while I was more interested in migrating[3] content than anything else. And to be honest, Hugo[4] makes such a task very simple.

The thing is, it's not just about uploading the resulting HTML files; I also have a search engine[5] that must be regenerated every time I add new content. Again, not the end of the world, but still.

So I went full DevOps with Hugo and Asciidoctor and PHP and Python and all the other things I need to publish this website, but no Kubernetes. For many people, DevOps and Kubernetes are so connected[6] to one another that it's hard to separate them. But the truth is that you can go "full DevOps" with any technology.

DevOps without Kubernetes

The steps I took were the following:

1. Encapsulate in a first container image all the tools required to build the website. It's a container image deriving from `docker.io/klakegg/hugo:0.101.0-ext-asciidoctor` with a few more bells and whistles.
2. Encapsulate in another container the search engine index generation. This one is a bit more complex, because I needed PHP and Python in it, but nothing to phone home about. Of course both containers are stored in GitLab's very convenient image registry.

[1] /blog/matomo/
[2] /blog/reboot/
[3] /blog/migrating-from-wordpress-to-hugo/
[4] https://gohugo.io/
[5] /blog/search-engine-for-akos.ma/
[6] https://deprogrammaticaipsum.com/antonomasia/

```
# Install PHP dependencies
FROM docker.io/library/composer:2.5.5 AS composer
WORKDIR /build
COPY composer.json /build/composer.json
RUN composer install

# Runtime image
FROM docker.io/library/ubuntu:22.04
ENV DEBIAN_FRONTEND=noninteractive
RUN apt update && apt install -y sqlite php8.1 php8.1-sqlite3 php8.1-mysql php8
WORKDIR /build

# Install Python dependencies
COPY requirements.txt /build/requirements.txt
RUN pip3 install -r requirements.txt

COPY --from=composer /build/vendor /build/vendor
COPY src /build/src
```

3. Create a `.gitlab-ci.yml` file for this website putting all of these tools in action[7]:

```
stages: [build, index, deploy]

hugo:
  stage: build
  image:
    name: registry.gitlab.com/akosma/hugo:1.2
    entrypoint: [""]
  cache:
    policy: pull-push
    paths:
      - build/public
  script:
    - hugo --minify --quiet --baseURL https://akos.ma --destination build/publi
  artifacts:
    paths:
      - build/public
  only:
    - schedules

index:
  stage: index
  image:
    name: registry.gitlab.com/akosma/search-engine-for-akos.ma:1.5
    entrypoint: [""]
  cache:
    policy: pull-push
```

[7] The value of the `SSH_HOST_KEY` can be found with the `ssh-keygen -H -F hostname` command.)

```
    paths:
      - build/public
      - build/index
  script:
    - mkdir -p build/index
    - rm build/index/index.db
    - php /build/src/create_index.php build/public build/index
    - python3 /build/src/create_files_db.py build/public build/index
  artifacts:
    paths:
      - build/index
  only:
    - schedules

deploy:
  stage: deploy
  image: docker.io/library/alpine:3.17.2
  cache:
    policy: pull
    paths:
      - build/public
      - build/index
  before_script:
    - apk update && apk add openssh-client rsync
    - eval $(ssh-agent -s)
    - echo "$SSH_PRIVATE_KEY" | ssh-add -
    - mkdir ~/.ssh
    - echo "$SSH_HOST_KEY" > ~/.ssh/known_hosts
  script:
    - 'rsync --recursive --quiet --update ./build/public/* "$DEPLOYMENT_PATH_HT
    - 'scp ./build/index/index.db "$DEPLOYMENT_PATH_SEARCH"'
  only:
    - schedules
```

4. Create a scheduled pipeline[8] in GitLab, to be executed every Friday morning at 08:00 AM (Zürich time).

Thanks to containers, I can reuse them[9] on the `Makefile` that I use to generate the website, and even better, I don't need to install PHP, Python, or even Hugo. My only requirement is Podman[10] or Docker.

```
PODMAN := podman run --rm --volume "$${PWD}":/build
IMAGE_HUGO := registry.gitlab.com/akosma/hugo:1.2
HUGO := $(PODMAN) --entrypoint="hugo" $(IMAGE_HUGO) --minify --quiet
PREVIEW := $(PODMAN) --publish 1313:1313 $(IMAGE_HUGO)

IMAGE_INDEXER := registry.gitlab.com/akosma/search-engine-for-akos.ma:1.5
SEARCH_INPUT := $(CURDIR)/build/public
```

[8] https://docs.gitlab.com/ee/ci/pipelines/schedules.html
[9] /blog/reusing-apps-between-teams-and-environments-through-containers/
[10] https://podman.io/

```
SEARCH_OUTPUT := $(CURDIR)/build/index

.PHONY: all
all: html

.PHONY: preview
preview:
    $(PREVIEW) server --bind "0.0.0.0" --buildDrafts --buildFuture --baseURL ht

.PHONY: html
html:
    $(HUGO) --baseURL https://akos.ma --destination build/public

.PHONY: search_index
search_index: build/index $(SEARCH_OUTPUT)/index.db

build/index:
    mkdir -p build/index

$(SEARCH_OUTPUT)/index.db:
    podman run --rm --volume $(SEARCH_INPUT):/build/input --volume $(SEARCH_OUT
    podman run --rm --volume $(SEARCH_INPUT):/build/input --volume $(SEARCH_OUT

.PHONY: clean
clean:
    rm -rf build
```

Deployment

As you can see in the YAML above, the deployment process involves good old `rsync` and `scp`, and the pipelines only work when triggered by a schedule. The inspiration of this task comes directly from this page[11].

Et voilà. Full DevOps with traditional web hosting servers. Just `git push` articles for future editions during the week, and let GitLab wake up and do its job automatically for you on Friday morning.

[11] https://www.howtogeek.com/devops/how-to-use-rsync-and-ssh-in-a-dockerized-gitlab-ci-pipeline/

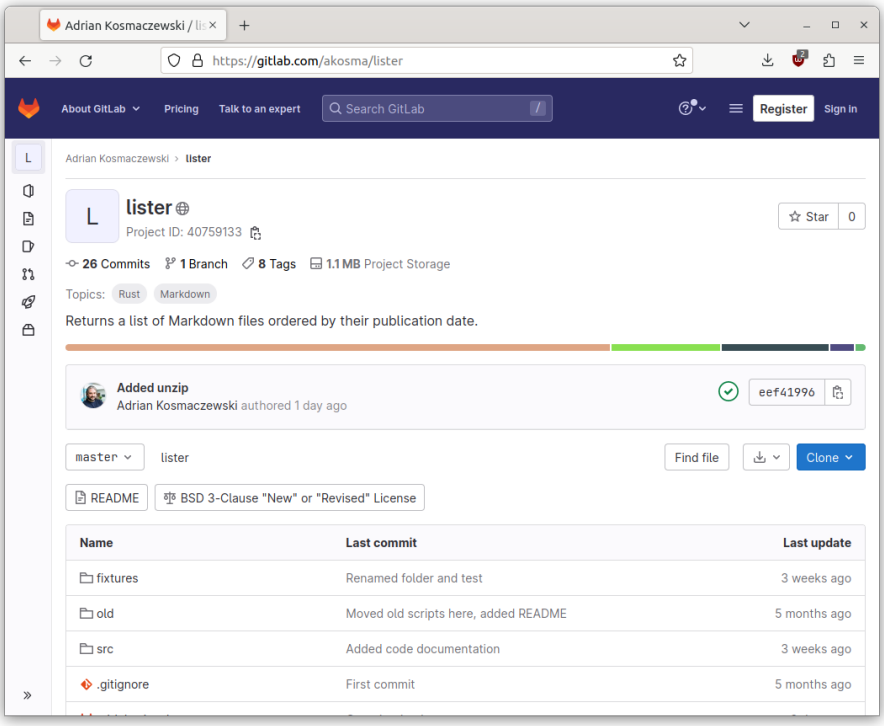
Exporting Hugo to PDF

Adrian Kosmaczewski

2023-04-28

Hugo[1] is fantastic but it misses one key functionality: **the generation of PDF files.** This article provides a possible solution for it using Podman, Pandoc, and a custom tool built in Rust.

The Rust project is called lister[2] and is available in GitLab for you to use and extend. It contains a small utility called `lister` that does only one thing: it lists all Hugo source files in Markdown format and returns a list of files by ascending publication date. This means that the tool opens each file, reads the `date:` parameter on top, and uses that information to order the files.



[1] https://gohugo.io/
[2] https://gitlab.com/akosma/lister/

Container Image

The container image is available at `registry.gitlab.com/akosma/lister:1.7`. The `Dockerfile` below shows how it's built, with a 2-step process, first building the Rust executable, and then creating an image with Pandoc[3], TeX Live[4] with XeTeX[5], Poppler[6], git-restore-mtime[7][8], and some other tools.

```
# Step 1: build Rust tool
FROM docker.io/library/rust:alpine as builder
RUN apk update && apk add --no-cache openssl-dev musl-dev

# Compile dependencies
RUN USER=root cargo new --bin lister
WORKDIR ./lister
COPY ./Cargo.lock ./Cargo.lock
COPY ./Cargo.toml ./Cargo.toml
RUN cargo build --release

# Compile the app itself
RUN rm src/*.rs
RUN rm ./target/release/deps/lister*
ADD src ./src
ADD fixtures ./fixtures
RUN cargo test
RUN cargo build --release

# Step 2: runtime image
FROM docker.io/pandoc/core:3.1.1.0-ubuntu
ENV DEBIAN_FRONTEND noninteractive
RUN apt update && apt install -y texlive texlive-xetex \
     poppler-utils make git-restore-mtime curl unzip
WORKDIR /build
COPY --from=builder /lister/target/release/lister /usr/local/bin/lister
```

The compilation of Rust code can be quite long. This is why the `Dockerfile` is built in such a way that, when used in your own laptop, the dependencies will only be downloaded once and cached. This speeds up the process of debugging and creation until you're ready to ship, wink wink.

[3] https://pandoc.org/

[4] https://tug.org/texlive/

[5] https://en.wikipedia.org/wiki/XeTeX

[6] https://en.wikipedia.org/wiki/Poppler_(software)

[7] https://github.com/MestreLion/git-tools

[8] The `git-restore-mtime` tool is there to reuse the `Makefile` in a CI/CD pipeline, for example GitLab, avoiding the rebuilding of all files by first setting the dates of files (used by `make` to keep track of things) to that of their last modification.

Makefile

Here's the `Makefile` that drives the creation of those PDFs using the `lister` container image:

```
PODMAN := podman run --rm --volume "$${PWD}":/build
IMAGE_LISTER := registry.gitlab.com/akosma/lister:1.7
PANDOC ?= $(PODMAN) --entrypoint "pandoc" $(IMAGE_LISTER)
PDFUNITE ?= $(PODMAN) --entrypoint "pdfunite" $(IMAGE_LISTER)
LISTER ?= $(PODMAN) --entrypoint "lister" $(IMAGE_LISTER)
SOURCES := $(shell $(LISTER) content/posts)
OBJECTS := $(SOURCES:content/posts/%/index.md=build/pdf/%.pdf)

.PHONY: all
all: fullpdf

.PHONY: clean
clean:
    rm -rf build

.PHONY: debug
debug:
    @echo $(SOURCES)

.PHONY: fullpdf
fullpdf: build/pdf build/pdf/_full_website_archive.pdf

build/pdf:
    mkdir -p build/pdf

build/pdf/%.pdf: content/posts/%/index.md
    $(PANDOC) --write=pdf --pdf-engine=xelatex \
        --variable=papersize:a4 --variable=links-as-notes \
        --variable=mainfont:DejaVuSans \
        --variable=monofont:DejaVuSansMono \
        --resource-path=$$(dirname $<) --out=$@ $< 2> /dev/null

build/pdf/_full_website_archive.pdf: $(OBJECTS)
    @echo "Building _full_website_archive.pdf"
    @$(PDFUNITE) scripts/pdf_cover.pdf $^ $@
```

The `make debug` command shows the contents of the `SOURCES` variable, with the list of Markdown files ordered by publication date, excluding those in the future.

If the source file of this article is located in the path `content/posts` (as is the case with Hugo) you can use this `Makefile` as follows:

```
$ make build/pdf/exporting-hugo-to-pdf.pdf
```

The important thing in the `Makefile` above is that the PDF files will be regenerated only if their corresponding Markdown file was updated.

The filename of the PDF file will automatically correspond to the slug of its source article.

Finally, we concatenate all PDF files into a single one called `_full_website_archive.pdf` using the `pdfunite` tool of the Poppler[9] package[10]. I added a `pdf_cover.pdf` file made with LibreOffice just for the sake of completeness.

To show the product of these scripts in action, you can download the PDF version[11] of this article. The file features links as footnotes, images, links, formatting, and respects the source code highlighting of the code snippets.

[9] https://en.wikipedia.org/wiki/Poppler_(software)

[10] At the time of this writing, the final PDF file, including all of the articles in this blog, takes around 30 minutes to build on my 2019 ThinkPad Lenovo X1 Carbon, and has... almost 1800 pages!

[11] exporting-hugo-to-pdf.pdf

The Oldest Web Pages I've Made Still Online

Adrian Kosmaczewski

2023-05-05

Exploring old backups I came across the links to the oldest websites that I've made, between 1997 and 1998, as part of my work at FIS[1], 25 years ago.

(And this discovery happened as the Web celebrated its 30th anniversary[2] last Sunday!)

The URLs of those dinosaur pages haven't changed since (which is the most surprising part of the story!) and in some cases my name appears in the "Author" field (something my boss at the time didn't appreciate at all, because... reasons):

```
<META NAME="Author" CONTENT="Adrian Kosmaczewski">
```

Another <META> tag gives an idea of the tooling used to create these early websites from the pre-Dot-com boom[3] era:

```
<META NAME="GENERATOR" CONTENT="Mozilla/3.03Gold (Win95; I) [Netscape]">
```

The full list of the oldest URLs still online that have my signature is this:

- http://www.fis-net.com/invertec/ (1997) made with the Netscape[4] 3 web page editor
- http://www.fis-net.com/azcona/ (1997) made with FrontPage[5] 2.0
- http://www.fis-net.com/legaspi/ (1997) made with FrontPage 3.0
- http://www.fis-net.com/fujian/ (1998) made with FrontPage 3.0
- http://www.fis-net.com/hunan/ (1998) made with FrontPage 3.0
- http://www.fis-net.com/haikui/ (1998) made with FrontPage 3.0
- http://www.fis-net.com/expopesca/ (1998) made with Macromedia Fireforks[6]
- http://www.fis-net.com/guinea/ (1999) made with Macromedia Fireworks

[1] /blog/my-job-at-fis-international-co.-ltd./
[2] https://home.cern/news/news/computing/30-years-free-and-open-web
[3] https://en.wikipedia.org/wiki/Dot-com_bubble
[4] https://en.wikipedia.org/wiki/Netscape_(web_browser)
[5] https://en.wikipedia.org/wiki/Microsoft_FrontPage
[6] https://en.wikipedia.org/wiki/Adobe_Fireworks

- http://www.fis-net.com/dat-schaub/ (1999) made with Macromedia Fireworks

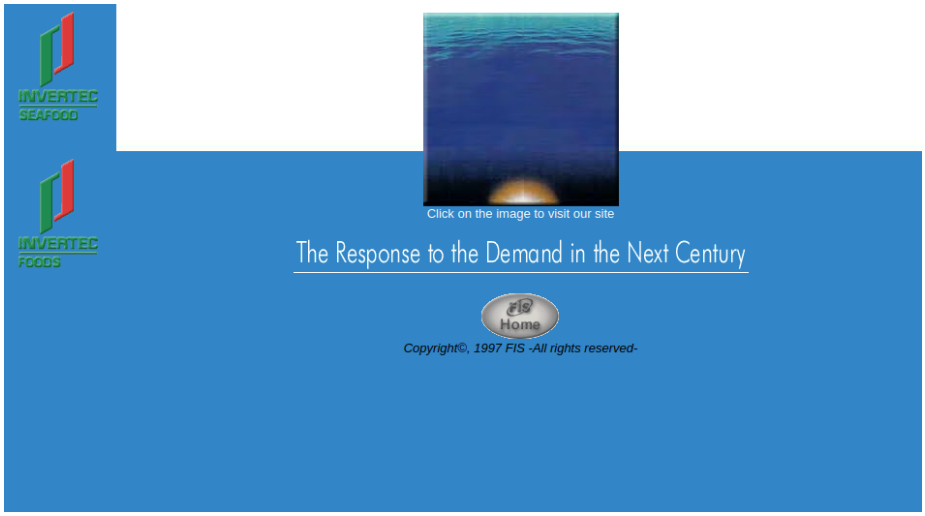
These websites give an idea of the processes and tooling required to be a “frontend engineer” back in 1997 (we called themselves “web developers” back then, kids.)

- UPPERCASE HTML TAGS.
- Lots of frames; not <IFRAME> but good old <FRAMESET>.
- Not a lot of CSS at the beginning, then a bit more, particularly for colors.
- <FONT> tags to control the (admittedly limited) typography choices.
- Lots of GIFs, animated or not (these pages have a strong GeoCities[7] vibe, let’s be honest.)
- A little bit of JavaScript here and there.
- No UTF-8; instead:

```
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
```

- No HTML 5; instead:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 3.2//EN">
```



Finding these websites was quite a surprise and a trip down memory lane. I hope they won’t be removed now that I’ve published this blog post! Since the Internet Archive has not saved these pages, I’ve saved a copy using the `wget -mpEk "http://…"`[8] command, because you never know. The archive is right here[9].

Update, 2023-05-12: Found two more: a paper published for a friend around February 1997 (Internet Archive[10] snapshot taken

[7] https://en.wikipedia.org/wiki/Yahoo!_GeoCities
[8] https://superuser.com/a/1673816
[9] old-websites-archive.zip
[10] https://web.archive.org/web/20001108235800/http://uni2a.unige.ch/~barras3/

November 8th, 2000) and a “do-it-yourself home page” website maker application I wrote for FIS (Internet Archive[11] snapshot taken May 16th, 2001).

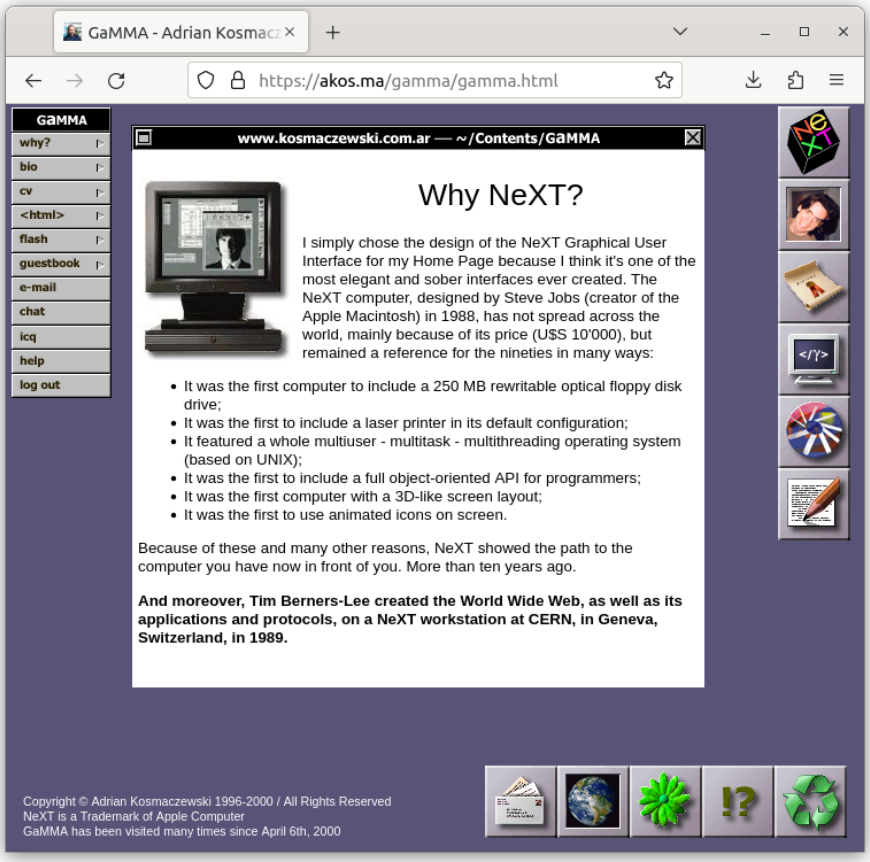
[11] https://web.archive.org/web/20010516210455/http://pages.fis.com/do-it/

GaMMA

Adrian Kosmaczewski

2023-05-12

Digging in my archives I found a backup of my personal home page from 2000 to 2003, and through a little work of archeology and restoration, I made it work in our modern world of 2023.



Back in those days I was starting my exploration of computer history, and the NeXT computer[1] appeared everywhere as one of the major inspirations of our modern computing experience. Hence I decided to

[1] https://en.wikipedia.org/wiki/NeXT_Computer

create **GaMMA**[2], a website I described as "a DHTML[3] experience".

This ancestor of single-page applications[4] emulates a NeXT desktop environment[5], where people could click on the menu or on the dock icons, and then the content would dynamically load and appear on a draggable `<div>` object. This was 5 years before AJAX[6] was born!

How did this work? Well, I used a hidden `<iframe>` to load children pages, who would call some JavaScript on the parent page to indicate that they had finished loading. And it worked!

Back to the Future

But running GaMMA in our current world was a bit problematic:

- It was created in a world with Internet Explorer 4 and 5, and Netscape 4; there were literal booleans testing whether you were using either of these browsers, and Firefox wouldn't be there for 5 more years:

```
ns4 = (document.layers)? true:false
ie4 = (document.all)? true:false
```

- The backend consisted of Classic ASP[7] pages written in VBScript[8] (that's what I used for work, after all) that stored data in a set of Microsoft Access[9] databases (I know, I know, a really poor choice. Anywaaaaaaaay.)
- I made the whole website with Macromedia Dreamweaver[10] (it wasn't yet an Adobe product,) which meant that the JavaScript inside was minified, unreadable… and incompatible with 2023.
- And to top it all, GaMMA featured some of my Flash[11] movies!

So how did I restore this website?

1. Removing Server-Side Code

I started by the easiest part, removing all the VBScript and Classic ASP code. I thought about rewriting that in PHP, but it was just used

[2] The name "GaMMA" means it was my third personal homepage, and the lowercase "a" is a nod to the "e" in NeXT.

[3] https://en.wikipedia.org/wiki/Dynamic_HTML

[4] https://en.wikipedia.org/wiki/Single-page_application

[5] Be mindful that I had never used a NeXT computer at that moment, so my recreation of the UI and its interactivity was based on my perceptions after seeing screenshots found all over the web. It was not meant as a faithful recreation, but more as an exercise in style, and a technical challenge in web development.

[6] https://en.wikipedia.org/wiki/Ajax_(programming)

[7] https://en.wikipedia.org/wiki/Active_Server_Pages

[8] https://en.wikipedia.org/wiki/VBScript

[9] https://en.wikipedia.org/wiki/Microsoft_Access

[10] https://en.wikipedia.org/wiki/Adobe_Dreamweaver

[11] https://en.wikipedia.org/wiki/Adobe_Flash

to store information about the number of visitors and to read and write data in the guestbook (remember guestbooks[12]?)

So instead I just stripped all of that server code away. And I removed the Microsoft Access databases from the equation, too. I renamed the files from `*.asp` to `*.html`.

2. Replacing Dreamweaver-Generated Code

I had to rewrite and replace quite a bit of the client-side JavaScript code. First I decided to simplify those booleans at the top of the script:

```
//ns4 = (document.layers)? true:false
//ie4 = (document.all)? true:false
ns4 = true
ie4 = false
```

Yes, I use Firefox, which it is technically a direct heir to Netscape, so `ns4 = true` and voilà.

The biggest part of the JavaScript code was automatically generated by Dreamweaver, and it depended on either the `document.layers` property of Netscape or the `document.all` property of Internet Explorer… both of which are deprecated today.

But this is important: both were dictionaries, not functions, and both roughly worked like the `document.getElementById()` function nowadays. So the trick was to use an ES2015 `Proxy` object to create the polyfill of the year:

```
// Courtesy of
// https://stackoverflow.com/a/20147219/133764
document.layers = new Proxy({}, {
    get: function(target, name, receiver) {
        if (!(name in target)) {
            return document.getElementById(name);
        }
    }
});
```

There's also an autogenerated function called `MM_findObj()` used by Dreamweaver, quite literally doing what `document.getElementById()` does nowadays, so:

```
/*
function MM_findObj(n, d) { //v3.0
    var p,i,x;  if(!d) d=document; if((p=n.indexOf("?"))>0&&parent.frames.lengt
    d=parent.frames[n.substring(p+1)].document; n=n.substring(0,p);}
    if(!(x=d[n])&&d.all) x=d.all[n]; for (i=0;!x&&i<d.forms.length;i++) x=d.for
    for(i=0;!x&&d.layers&&i<d.layers.length;i++) x=MM_findObj(n,d.layers[i].doc
}
*/
```

[12] https://en.wikipedia.org/wiki/Guestbook

```
function MM_findObj(n, d) {
    return document.getElementById(n);
}
```

Other functions, like `MM_showHideLayers()`, worked without problems; some parts of the standards we use today haven't changed in 24 years.

3. Dynamically Loading Data

If you click on menus or dock items, the page loads its contents dynamically from the server. In the original GaMMA, there were two APIs at play:

- In Internet Explorer 4 or 5, one had to set the `url` property;
- In Netscape 4, one would use the `load()` method instead:

```
//Load external HTML file in specific layer
function loadSource(id,nestref,url) {
    if(!loadingRightNow) {
        loadingRightNow = true;
        MM_showHideLayers('loading','','show');
        if (ns4) {
            var lyr = (nestref)? eval('document.'+nestref+'.document.'+id) : do
            lyr.load(url,lyr.clip.width)
        }
        else if (ie4) {
            parent.bufferFrame.document.location = url
        }
        loadingRightNow = false;
    }
}
```

(Pay attention to the `loadingRightNow` variable and its feeble attempts to prevent this code from stepping on its own feet.)

When the child page had loaded into the `<iframe>`, it would call the `loadSourceFinish()` function on the parent, which is something only Internet Explorer needed.

```
//End of External HTML loading
function loadSourceFinish(id) {
    if (ie4) document.all[id].innerHTML = parent.bufferFrame.document.body.inne
    MM_showHideLayers('loading','','hide');
    MM_showHideLayers('winMini','','hide','win','','show','contents','','show')
}
```

But we're in 2023, and nowadays such an operation is trivial, thanks to the `fetch()` API and the `async` and `await` keywords, coupled with the `DOMParser` API and its capacity to parse content from text strings:

```
async function loadSource(id, nestref, url) {
    MM_showHideLayers('loading','','show')
```

```
    const data = await fetch(url)
    const html = await data.text()
    let parser = new DOMParser()
    let doc = parser.parseFromString(html, "text/html")
    document.win.innerHTML = doc.querySelector('body').innerHTML
    MM_showHideLayers('loading','','hide')
    MM_showHideLayers('winMini','','hide','win','','show','contents','','show')
}
```

Gotta love being a web developer in 2023. Even the semicolons are gone. And let's be honest, the snippet above feels a lot like HTMX[13] or Hotwire[14] at work.

4. Drag-and-Drop

Let us now talk about the horrendous `MM_dragLayer()` function that Dreamweaver inserted in pages to enable drag and drop of `<div>` and `<layer>` objects in 1999. I'm including here the first few lines just for historical purposes:

```
function MM_dragLayer(objName,x,hL,hT,hW,hH,toFront,dropBack,cU,cD,cL,cR,targL,
  //Copyright 1998 Macromedia, Inc. All rights reserved.
  var i,j,aLayer,retVal,curDrag=null,NS=(navigator.appName=='Netscape'), curLef
  if (!document.all && !document.layers) return false;
  retVal = true; if(!NS && event) event.returnValue = true;
  if (MM_dragLayer.arguments.length > 1) {
    curDrag = MM_findObj(objName); if (!curDrag) return false;
    if (!document.allLayers) { document.allLayers = new Array();
      with (document) if (NS) { for (i=0; i<layers.length; i++) allLayers[i]=la
        for (i=0; i<allLayers.length; i++) if (allLayers[i].document && allLaye
          with (allLayers[i].document) for (j=0; j<layers.length; j++) allLayer
      } else for (i=0;i<all.length;i++) if (all[i].style&&all[i].style.position
    curDrag.MM_dragOk=true; curDrag.MM_targL=targL; curDrag.MM_targT=targT;
// …
```

Yes, it's minified and everything. And, yeah, the complete function is about 5 times the length of the snippet above. Good luck understanding what's going on.

(Kids, this was 7 years before jQuery[15], Prototype[16], script.aculo.us[17], and even 5 years before Firebug[18] introduced `console.log()` to the world. Back in those days we could only `alert()` our way into code.)

Instead, today we would add the `draggable="true"` attribute to the `<div>`s we want to move, and then use the Drag & Drop API[19]. I found

[13] https://htmx.org/
[14] https://hotwired.dev/
[15] https://jquery.com/
[16] http://prototypejs.org/
[17] http://script.aculo.us/
[18] https://en.wikipedia.org/wiki/Firebug_(software)
[19] https://developer.mozilla.org/en-US/docs/Web/API/HTML_Drag_and_Drop_API

this code[20] to make things even simpler:

```
function dragElement(elmnt) {
  var pos1 = 0, pos2 = 0, pos3 = 0, pos4 = 0, elementMoving;
  if (document.getElementById(elmnt.id + "header")) {
    // if present, the header is where you move the DIV from:
    document.getElementById(elmnt.id + "header").onmousedown = dragMouseDown;
  } else {
    // otherwise, move the DIV from anywhere inside the DIV:
    elmnt.onmousedown = dragMouseDown;
  }

  function dragMouseDown(e) {
    e = e || window.event;
    e.preventDefault();
    // get the mouse cursor position at startup:
    pos3 = e.clientX;
    pos4 = e.clientY;
    document.onmouseup = closeDragElement;
    // call a function whenever the cursor moves:
    document.onmousemove = elementDrag;
  }

  function elementDrag(e) {
    e = e || window.event;
    e.preventDefault();
    // calculate the new cursor position:
    pos1 = pos3 - e.clientX;
    pos2 = pos4 - e.clientY;
    pos3 = e.clientX;
    pos4 = e.clientY;
    // set the element's new position:
    elmnt.style.top = (elmnt.offsetTop - pos2) + "px";
    elmnt.style.left = (elmnt.offsetLeft - pos1) + "px";
  }

  function closeDragElement(event) {
    // stop moving when mouse button is released:
    document.onmouseup = null;
    document.onmousemove = null;
    if (elmnt.id === 'winMini') {
      repositionWinBar('winMini', 'win');
    }
    if (elmnt.id === 'win') {
      repositionWinBar('win', 'winMini');
    }
  }
}
```

[20] https://www.w3schools.com/howto/howto_js_draggable.asp

The new drag-and-drop code is not only easier to read, it is also much more flexible, allowing users to drag and drop not only by holding the title bar, but any part of the `<div>` they want.

5. Flash!

And here comes the most incredible part. Back in those days I produced a lot of Macromedia Flash[21] movies, and I still have plenty of those `.swf` files around. The problem is... well, Flash is completely gone[22]:

> Since Adobe no longer supports Flash Player after December 31, 2020 and blocked Flash content from running in Flash Player beginning January 12, 2021, Adobe strongly recommends all users immediately uninstall Flash Player to help protect their systems.

Those movies are lost. Or are they? Enter the Ruffle[23] project. It is a little marvel of modern technology: written in Rust, it generates WebAssembly code that emulates a Flash player. You read right. And to prove it, I've re-enabled the Flash movies page in GaMMA completely[24].

One More Thing

Actually the original GaMMA included a live web chat application, using a Microsoft Access database (again) and live-reloading messages in spite of the fact that the WebSockets[26] and Server Sent Events[27] APIs were literally decades away.

How did it work? Well, it simply refreshed the page periodically:

```
<META HTTP-EQUIV="Refresh" CONTENT="30; URL=bottom.asp">
```

People would write messages on the `top` part of the `<frameset>` and the bottom would refresh itself every 30 seconds. Boom, you have yourself a chat. Maybe I'll rewrite it in the future, but not now.

Conclusion

No wonder my introduction in the GaMMA website reads:

> I hope this will be the last HTML page I'll ever do... even if this one is more of a JavaScript-driven one... Maybe "Delta" will be a Flash website?

[21] https://en.wikipedia.org/wiki/Adobe_Flash

[22] https://www.adobe.com/products/flashplayer/end-of-life.html

[23] https://ruffle.rs/

[24] I'll write more about Ruffle and Flash in an upcoming article[25].

[26] https://developer.mozilla.org/en-US/docs/Web/API/WebSockets_API

[27] https://developer.mozilla.org/en-US/docs/Web/API/Server-sent_events/Using_server-sent_events

Famous last words. Actually Flash went the way of the Dodo, and thankfully web standards evolved in wonderful ways. Yes, it's still a PITA to create websites, but when I look at the past, I'm thankful for the many wonderful APIs[28] we have on our browsers these days.

GaMMA is available in this very website[29] for your viewing and clicking pleasure.

[28] https://caniuse.com/
[29] /gamma/

Back to Monoliths

Adrian Kosmaczewski

2023-05-19

So Amazon Prime Video (of all people!) published a blog post[1] about how they're returning to monoliths, relayed by DHH[2], generating lots of noise, to the point that even Dr. Werner Vogels himself, CTO at Amazon, had to pour some thoughts[3] about the subject.

Monoliths are back, baby! Just like dynamically-typed languages return every so often[4] after a decade or so of strongly-typed languages. These are waves that come and go because our craft is young, and we're still figuring things out.

And also because of ageism: lots of teams lose their older team members, and companies filled with junior (read: inexperienced) developers tend to repeat the mistakes of the past.

As far as I'm concerned I'm a big fan of monoliths, just like I'm a big fan of server-side rendering[5] (another trend that is slowly returning.)

In my first job[6] we had a single Pentium Pro[7] server running both IIS[8] and SQL Server[9] with Windows NT 4.0[10], with a monolithic ASP 1.0[11] application on top written in VBScript[12], serving hundreds of thousands of visitors per month without any caching strategy.

Let me repeat that. Tens of thousands of visitors per day, on a single-core Pentium Pro, running both the web server and the database engine, without any cache (which means that all requests hit the database, all the time.)

In 2000 we moved to a two-servers solution based on Pentium III[13]

[1] https://www.primevideotech.com/video-streaming/scaling-up-the-prime-video-audio-video-monitoring-service-and-reducing-costs-by-90

[2] https://world.hey.com/dhh/how-to-recover-from-microservices-ce3803cc

[3] https://www.allthingsdistributed.com/2023/05/monoliths-are-not-dinosaurs.html

[4] https://deprogrammaticaipsum.com/the-truce-of-type-inference/

[5] /blog/server-side-rendering-ftw/

[6] /blog/my-job-at-fis-international-co.-ltd./

[7] https://en.wikipedia.org/wiki/Pentium_Pro

[8] https://en.wikipedia.org/wiki/Internet_Information_Services

[9] https://en.wikipedia.org/wiki/Microsoft_SQL_Server

[10] https://en.wikipedia.org/wiki/Windows_NT_4.0

[11] https://en.wikipedia.org/wiki/Active_Server_Pages

[12] https://en.wikipedia.org/wiki/VBScript

[13] https://en.wikipedia.org/wiki/Pentium_III

CPUs (one for the web server, another for the DB) both running Windows 2000 Advanced Server[14]. Still no cache. Same ASP code. Blazingly fast performance.

Whether microservices are a good choice for your architecture or not depends on two factors:

1. Your team structure[15].
2. Being Netflix or not.

In all other cases, I agree with DHH; monoliths are fantastic choices for small teams, and these days, the following architecture might prove effective for 99.99% of all your needs:

1. An application server (most probably in C#, Crystal[16], or Go);
2. A database server (most probably PostgreSQL[17]);
3. A messaging server (to send lots of email, or to act as a proxy to an SMS or other kind of external messaging system);
4. A cache server (most probably Redis[18]);
5. A message queue (with RabbitMQ[19] or Kafka[20] or similar) in the middle.

And that's it. Whether these are physical or virtual machines or containers or pods does not matter. It literally doesn't. Just five things to manage, just five things to update, just five things to debug.

Now, we're in 2023, and we have great tools at our disposal for literally zero dollars: containers, Kubernetes, database engines, messaging queues, and whatnot. But most of all, we've got more experience about running web apps today than in 1997.

So the important thing is to make your app follow the 12 Factor Principles[21], and in particular, make it configurable via environment variables, so that your developers can run your monolith locally in development, and your operations can configure it properly in production.

Use the best design patterns inside your application[22], so that you could eventually break it up in pieces in the future; yes, for example programming against interfaces, and injecting services at runtime (something ASP.NET[23] and Quarkus[24] do wonderfully and off-the-box, to name a few frameworks.)

[14] https://en.wikipedia.org/wiki/Windows_2000

[15] /blog/microservices-or-not-your-team-has-already-decided/

[16] /blog/crystal-is-a-surprise/

[17] https://www.postgresql.org/

[18] https://redis.io/

[19] https://www.rabbitmq.com/

[20] https://kafka.apache.org/

[21] https://12factor.net/

[22] I don't care what people say: object-oriented code, when done thoughtfully, is a wonderful thing. That usually means less inheritance and more composition, and applying a few techniques from the functional world, like immutable variables.

[23] https://asp.net/

[24] https://quarkus.io/

So if your app becomes so successful that you must expand your team, make your monolith capable of being broken thanks to good old interfaces and dependency injection.

Having a monolith will make your app infinitely more simple to design (less moving parts), debug (just launch it with the proper environment variables and plug your favorite debugger), and deploy (just one instance to run.)

Using server-side rendering will also make your app simpler. Just defer your logic to the server. Make the client as simple as possible. Mustache templates[25] are your friend.

Use an ORM such as Entity Framework Core[26], Doctrine[27], GORM[28], or Active Record[29] to make your database code as simple as possible, and your migrations more explicit and easier to understand.

As for updates, forget about canary deployments[30] and stuff like that; show a temporary banner saying that your app is in maintenance, migrate your database, upgrade the app version, and remove the banner. You're not Netflix, remember, which means you can most probably afford a few minutes of downtime.

Build containers[31] with your monolith inside. Make them as small as possible (Alpine[32] is your friend, but beware of musl[33]), and configurable via environment variables. In most cases you won't need more than one instance in production (no load balancing, really) and you can even run it locally, or on a staging environment, or on a Kubernetes cluster in staging. Yes, Kubernetes[34] can be a great choice for running your monolith; all those YAML files are a great way to document your deployments and variables and all that's needed to automate your life.

If you build containers for your system, ensure they don't expose ports lower than 1024, and that they don't run as `root`. If you ever have to run them on OpenShift[35] you'll be glad you followed this advice. Podman[36] and OpenShift Local[37] are your friends in this area.

Needless to say, use a CI/CD system. Your team should be able to deploy in production a few times a week without much effort. The

[25] https://mustache.github.io/
[26] https://learn.microsoft.com/en-us/ef/core/
[27] https://www.doctrine-project.org/
[28] https://gorm.io/
[29] https://guides.rubyonrails.org/active_record_basics.html
[30] https://learn.microsoft.com/en-us/azure/devops/pipelines/ecosystems/kubernetes/canary-demo?view=azure-devops&tabs=yaml
[31] /blog/reusing-apps-between-teams-and-environments-through-containers/
[32] https://www.alpinelinux.org/
[33] https://musl.libc.org/
[34] /blog/kubernetes-for-non-technical-readers/
[35] https://www.redhat.com/en/technologies/cloud-computing/openshift
[36] https://podman.io/
[37] https://developers.redhat.com/products/openshift-local/overview

tooling these days (GitHub Actions[38], GitLab pipelines[39], Argo CD[40], Tekton[41], etc.) is seriously amazing. Remember to run your unit tests as part of the workflow.

Here I am advocating for a return to sanity. Let us all choose wisely, finding the best architecture for our systems; and yes, in many cases, monoliths will be a great choice, if not the best.

PS: And here's Kent Beck saying that waterfall is back[42]. More on this soon.

[38] /blog/reusing-apps-between-teams-and-environments-through-containers/
[39] https://docs.gitlab.com/ee/ci/pipelines/
[40] https://argoproj.github.io/cd/
[41] https://tekton.dev/
[42] https://www.youtube.com/watch?v=J4ihLROXzPk

Macromedia Flash

Adrian Kosmaczewski

2023-05-26

For about 4 or 5 years, roughly from 1999 to 2004, Macromedia Flash was a big part of my career. I started making Flash movies for fun around 1998, but by 1999 I was already making them as part of my day-to-day job. The founder of that company even used some of those movies to pitch to investors… successfully… so I get they worked fine after all.

Most of my work actually involved standard web technologies, but I did attend some Macromedia events in Buenos Aires; the Flash galaxy was strong in Argentina back then; comes to mind the now legendary (and 100% politically incorrect) webcomic "El Mono Mario"[1], entirely built with Flash.

After moving back to Europe, I made a few websites in Flash for some customers, but that was it. Around 2004 I stopped using Flash for my websites or those of my customers. I did get back to Flash for a short while in 2008, attempting to make screen savers for the Mac with Flash, something I wrote about on this blog[2] back then, but nothing else.

The Developer Experience

At a time when browsers didn't bundle a developer console (something that Joe Hewitt would invent as the Firebug[3] extension for Firefox) and when the saint trinity of CSS, HTML, and JavaScript (or even Java applets) didn't have any groundbreaking APIs such as SVGs or CSS animation, Flash offered an integrated development environment (which looked more like a non-linear video editor) for creating vector-based content that somehow was simultaneously cross-platform, widely available, and visually appealing.

The cross platform thing meant that the Flash player was available for the two big browsers of its time: - As a now infamous ActiveX[4] component for Internet Explorer, with the class ID `D27CDB6E-AE6D-11cf-96B8-444553540000`, and requiring the `<OBJECT>` HTML

[1] https://es.wikipedia.org/wiki/El_Mono_Mario

[2] /blog/screen-savers-for-the-mac-using-flash/

[3] https://web.archive.org/web/20091115094010/http://getfirebug.com/

[4] https://en.wikipedia.org/wiki/ActiveX

tag. - As a standard Netscape plugin handling the MIME type `application/x-shockwave-flash`, downloadable from the URL http://www.macromedia.com/shockwave/download/index.cgi?P1_Prod_Version=ShockwaveFlas now advertising the “End of Life for Adobe Shockwave,” and used with the <EMBED> tag.

At some point, I don’t remember exactly when, the ActionScript[5] language used by Flash became a dialect of JavaScript, which made it much more approachable for web developers.

But to be honest, even if the whole experience was decades beyond whatever W3C standard technology and Java applets could offer, it was frankly quite cumbersome to maintain Flash movies after their first development.

And to make things even worse, in a world where Google was rising to stardom… Flash movies weren’t really searchable until it was too late (around 2009)[6].

The Inspiration

How did I get so enthusiastic about Flash? One of the culprits was, without any doubt, the Balthaser Studios website, long gone since but somebody recorded a video[7] showing it in action. Back in 1999, all of us doing Macromedia Flash dreamt of repeating this experience. This was the golden standard.

The Impact

Flash was both loved and hated. As much as people hate to admit it, it played a crucial role in the development of newer standards.

Take, for example, YouTube. It literally started its life thanks to the video streaming capabilities of the Flash player, making video accessible to millions of users around the world. This was seven or eight years before HTML video became a possibility.

Arguably, without Flash, there might never have been a YouTube in 2005. No wonder Adobe absorbed Macromedia that year.

Flash was a so ubiquitous part of the standard browsing experience, that when we co-organized the first conference about the iPhone in Geneva in October 2008, one of the questions[8] in the audience was, when are Mobile Safari or the Android Browser (Chrome for Android wasn’t a thing yet) going to have Flash support? Could a project like Gnash[9] provide it?

[5] https://en.wikipedia.org/wiki/ActionScript

[6] Interestingly enough, HTML pages generated by the Flash IDE would include all the texts and links in the movie as comments, for crawlers to read and index.

[7] https://www.youtube.com/watch?v=GfLTw50q5Yo

[8] https://vimeo.com/2197598

[9] https://www.gnu.org/software/gnash/

The answer would come in 2010.

The End

Steve Jobs wrote in April 2010 a (now legendary) open letter called "Thoughts on Flash"[10] where he stated the various reasons (some technical, some based in security concerns, and some more related to platform reach and ownership) about the lack of Flash on iOS.

It was the stuff of legends, but it is said that Google also tried to bring Flash to the Android phones of that era (way less powerful than those of today) and failed.

The writing was (literally) on the wall. By 2011, Adobe started moving away from Flash, in particular open sourcing PhoneGap, a framework created by a Canadian company called Nitobi they had bought in 2009. PhoneGap became Apache Cordova, and together with responsive designs and SPAs, the race for the standard-compliant web-based mobile app was officially launched.

In January 2021[11], a "kill switch" in all Flash player plugins rendered them unusable, even though by that time, nobody cared anymore about the platform.

The Legacy

Unfortunately, the demise of Flash meant that there's a lot of content on the web that simply can't be watched anymore[12].

Well, almost. Thanks to the open source community, and the (partial) open sourcing of the SWF format[13], many projects saw the light, trying to replicate the Flash plugin with less security holes and better performance. Among those early efforts, come to mind Shumway[14] and Gnash[15].

Revival via Ruffle

But the winner nowadays is Ruffle[16], built with Rust and compiled into WebAssembly, providing a way for us to replay those old Flash movies from yesteryear, on today's hardware and software platforms.

As proof, here's one of the last websites I made with Flash in 2002 (for a customer in Geneva), running flawlessly and securely on your modern browser of 2023:

[10] https://en.wikipedia.org/wiki/Thoughts_on_Flash
[11] https://www.adobe.com/products/flashplayer/end-of-life.html
[12] https://edition.cnn.com/2021/09/10/tech/digital-news-coverage-9-11/index.html
[13] https://en.wikipedia.org/wiki/SWF
[14] https://en.wikipedia.org/wiki/Shumway_(software)
[15] https://en.wikipedia.org/wiki/Gnash_(software)
[16] https://ruffle.rs/

Ruffle is trivially simple to use[17]:

> Ruffle will automatically replace any old-style Flash embeds on websites with Ruffle elements. This is ideal for someone looking to preserve a legacy website with a minimum amount of work.

That means that the HTML required to display the movie above is literally the same I used in the original website 21 years ago (note the <EMBED> tag inside of the <OBJECT> tag, so that this code would work respectively in both Netscape and Internet Explorer!)

```
<script src="https://unpkg.com/@ruffle-rs/ruffle"></script>
<OBJECT classid="clsid:D27CDB6E-AE6D-11cf-96B8-444553540000"
 codebase="http://active.macromedia.com/flash2/cabs/swflash.cab#version=4,0,0,0
 <PARAM NAME=movie VALUE="qm.swf">
 <PARAM NAME=menu VALUE=false>
 <PARAM NAME=quality VALUE=high>
 <PARAM NAME=bgcolor VALUE=#000000>
 <EMBED src="qm.swf" menu=false quality=high bgcolor=#000000
 WIDTH=550 HEIGHT=400 TYPE="application/x-shockwave-flash"
 PLUGINSPAGE="http://www.macromedia.com/shockwave/download/index.cgi?P1_Prod_Ve
</OBJECT>
```

Here's another movie, used as a background animation during the presentation of the Martin Ennals Award[18], regularly updated from 2002 to 2007[19], and featuring all the winners and the sponsoring organizations involved in the award ceremony.

Here goes another Flash movie from 1999, a quick shot at a screensaver for my company–but you might want to turn the volume of your device down. You've been warned!

Very cheesy and corny, I know. But this was top-notch technology back in 1999, I tell ya.

Finally, an introduction to my GaMMA[20] website, also done in Flash (and it would be a good idea to turn your volume down):

And you know what's the most ironic part of all this? Ruffle works perfectly well on iOS and Android. The circle is closed.

[17] https://github.com/ruffle-rs/ruffle/wiki/Using-Ruffle#polyfill
[18] https://www.martinennalsaward.org/
[19] I used to be the webmaster of the Martin Ennals Foundation from 2002 to 2007.
[20] /blog/gamma/

Fedora 38

Adrian Kosmaczewski

2023-06-02

In December 4th, 2005, I published my first blog post[1] about Linux. I wrote it on Ubuntu 5.10 “Breezy” after installing it on my faithful iBook G3. Many years have passed, and I’ve become a full-time Linux user[2] now, having used no other operating system in the past 5 years.

But in the meantime Ubuntu has become annoying in various ways, so it was time for a change. Don’t get me wrong, it was a great distribution to start with, but after using it every day for a while, I realized I needed a bit more power in my hands. Ubuntu is simply fantastic for beginners. I guess I’m just not one anymore.

The things that have annoyed me the most in Ubuntu are the following:

- Lack of recent packages in its `apt` package manager, which means scouting the interwebz until you find that what you’re looking for is (hopefully) packaged as a `*.deb` file, or (tough luck) only available through `./configure && make && make install`... so good luck installing all the dependencies.
- Their knee-jerk reaction against Flatpak[3] and their insistence in using Snap[4] instead, an inferior package manager meant to lock users into the Ubuntu ecosystem, filled with annoying issues, and not offering anything else to the table.
- Related to the previous point, the fact that Firefox was installed by default as a Snap package in the latest versions of Ubuntu was the final straw.

I found that installing a whole operating system just to replace one of its core components (Snap) with another (Flatpak) was really stupid. I wanted to change to a distribution where this wasn’t an issue. So I settled for Fedora 38[5].

Of all distributions, why Fedora[6]? The other two distributions I’ve

[1] /blog/ubuntu/
[2] /blog/migrating-from-macos-to-linux/
[3] /blog/linux-package-manager-strategy/
[4] https://snapcraft.io/
[5] https://fedoramagazine.org/announcing-fedora-38/
[6] https://fedoraproject.org/

considered were Debian[7] and Arch[8]. But having played around with RHEL[9] at work I felt that I would feel comfortable using Fedora right away, and my perception was true. I also wanted a mainstream distribution with packages available for those software titles not available on Flatpak or `dnf`, as is the case with Debian `*.deb` packages; There's a lot of `*.rpm` packages out there for Fedora, which means that instead of `sudo dpkg -i package.deb` now I needed to `sudo dnf install package.rpm`.

The installation of Fedora 38 on my personal 6th generation ThinkPad was as simple as expected; fill a short form, click the install button, and wait for less than 5 minutes. Fedora 38 comes bundled by default with exactly what's needed to be immediately productive. And after a few weeks of use, I gotta say that I'm really pleased.

- It recognized and worked with 100% of all of my current hardware, no questions asked. This included my HP LaserJet printer **and** scanner, my Logitech webcam, my Lenovo ThinkPad dock, two screens, and more.
- Fedora 38 comes with Podman[10] built-in by default. Simply perfect for my container[11] needs.
- The `dnf` package manager has pretty much all the latest versions of everything I needed to install, including many programming languages. For the rest, in particular desktop apps, there's Flatpak.
- Speaking about which, Flatpak is installed by default, and since version 38, both the `fedora` and `flathub` remotes are available off-the-box, opening the door to lots of excellent software.
- Even better, Firefox is not a Snap package but installed in `/usr/bin`, as it should, ensuring all of its extensions[12] work properly.
- GNOME 44[13] is very slick and modern, feels very intuitive to use. I've added a few extensions[14] to customize it my way, just as I was doing with Ubuntu.
- `dnf` has some neat subcommands, such as `dnf swap`[15] to exchange a library by another, or `dnf provides`[16] to find out which package contains a particular program.

To be honest, the biggest issue I had was not directly related to Fedora 38 per se, but to GNOME 44: they have (for some reason) decided to get rid of taskbar icons, which means that I had to install the

[7] https://www.debian.org/
[8] https://archlinux.org/
[9] https://www.redhat.com/en/technologies/linux-platforms/enterprise-linux
[10] https://podman.io/
[11] /blog/reusing-apps-between-teams-and-environments-through-containers/
[12] /blog/firefox-extensions/
[13] https://foundation.gnome.org/2023/03/22/introducing-gnome-44/
[14] /blog/gnome-extensions/
[15] https://dnf.readthedocs.io/en/latest/command_ref.html#swap-command-label
[16] https://dnf.readthedocs.io/en/latest/command_ref.html#provides-command-label

AppIndicator and KStatusNotifierItem Support[17] GNOME extension to get them back. I suppose the new “background apps” feature was supposed to replace them, but in the current incantation it’s a buggy and incomplete thing, definitely not ready for production yet. I suppose they’ll fix it in future releases.

Another issue I had was related to Firefox not correctly displaying some video formats, which I solved by installing the non-free version of FFmpeg[18] on my machine. Since I had installed the free version first, I used the command `sudo dnf swap ffmpeg-free ffmpeg --allowerasing` to migrate from one to the other.

All in all I’m very satisfied with this change. Fedora 38 feels much more stable, fast, and more flexible than Ubuntu, to such a degree that I migrated my work laptop to Fedora 38 before flying to attend the Red Hat Summit in Boston[19] last week. It’s a fantastic productivity environment.

[17] https://extensions.gnome.org/extension/615/appindicator-support/
[18] https://ffmpeg.org/download.html#build-linux
[19] https://www.youtube.com/watch?v=AlxEDDkAiVg

Memories of WWDC 2008

Adrian Kosmaczewski

2023-06-09

Exactly 15 years ago, on Monday, June 9th, 2008, I published a blog post[1] with a picture taken in the big room of the Moscone conference center in San Francisco, waiting for Steve Jobs to introduce the iPhone 3G to the world at the annual Apple World Wide Developers' Conference 2008.

I had downloaded the SDK the day it was published[2] and I jumped 100% into it. I knew[3] that learning Objective-C since 2002 would pay off one day.

I then created a small script[4] to generate iCal files from the official schedule[5] in JSON format, something that got noticed by a few people like Jeff LaMarche and the long gone TUAW blog, as I told it in this article[6] of this blog.

We traveled to San Francisco[7] with Claudia, and the official t-shirt said it all: the event was all about iPhone apps.

(Credits: Xevio's t-shirt archive[8])

[1] /blog/i-was-there/
[2] https://twitter.com/akosma/status/767714079
[3] https://twitter.com/akosma/status/767728929
[4] https://web.archive.org/web/20080522173428/http://kosmaczewski.net/wwdc2008/
[5] https://web.archive.org/web/20080724150704/http://developer.apple.com/wwdc/schedules/
[6] /blog/wwdc-schedules-in-ical-format/
[7] /blog/sunny-wwdc/
[8] https://xevio.us/2017/02/t-shirt-archive/#jp-carousel-4704

That was the first WWDC that sold out[9], in 2 months. Starting 2014, WWDC became an "invitation only" event, where you sign up and you either are chosen to go or not. I attended the event four times, in 2008[10], 2009[11], 2010[12], and finally in 2012[13].

WWDC 2008 was the one with the t-shirts reading "OS X iPhone" because the name "iOS" didn't exist until 2010 (for proof, see the picture at the bottom of this article). This event happened before GitHub appeared on the map. I did not have an iPhone on my pocket yet, but a good old feature phone from Ericsson. I was a few weeks away from finishing my Master's degree[14] thesis, and I remember I was working on it on the plane to San Francisco.

I met lots of friends in that event, many of which I am still in contact with after all of these years (mostly through Mastodon[15], as they have almost all left Twitter behind, and rightfully so.)

I blogged right before[16] and right after[17] the keynote, sitting outside the main room on top of Moscone West conference center. The following day I had a pastrami sandwich[18] for lunch.

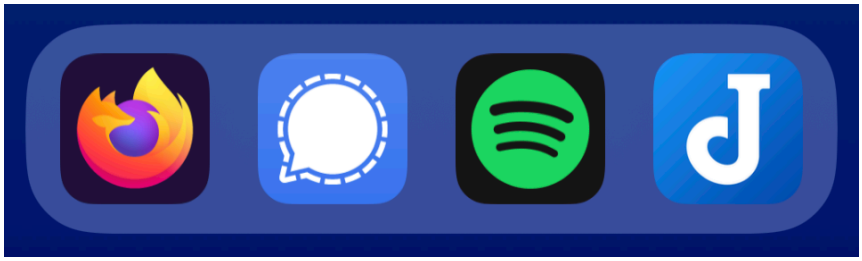
[9] https://twitter.com/akosma/status/327470684559249408
[10] /blog/wwdc-2008-keynote-main-points/
[11] /blog/wwdc-2009/
[12] /blog/wwdc-2010/
[13] /blog/attending-wwdc-2012/
[14] /blog/master/
[15] /blog/mastodon/
[16] /blog/waiting-for-the-keynote/
[17] /blog/wwdc-2008-keynote-main-points/
[18] /blog/pastrami-sandwich/

A few weeks later, an iPhone popped up on the Swiss Apple Store[19]. I had been expecting them for a year and a half[20] at that point.

After WWDC developers had to face the Fucking NDA[21] that blocked iPhone developers from asking questions on this newly created site called “Stack Overflow” and pretty much anywhere else. A ridiculous situation that only Apple could get out with. I asked[22] people to sign a petition[23] for Apple to drop it. I don’t know whether it worked or not, but by the end of September Apple did it.

While we were waiting for the NDA to drop we organized the one and only iPhone Conference in Geneva[24], held in October, for which we were awarded a “cease and desist” by Apple themselves (because the conference banner displayed… an iPhone). We literally turned the iPhone around in the picture (showing the camera instead of the screen) and did the conference anyway. The video of my talk[25] is still up there. I was younger and had darker hair. Same latino accent while speaking English, though.

15 years later, I have completed various other chapters of my life, and among other things, I have left Apple almost completely behind. These days I only use an iPhone that has a home screen that looks like this:



The toolbar at the bottom of my iPhone shows no Apple apps; well, maybe Firefox would qualify as one, as it’s just a decoration around the standard Safari browser engine, but still. No Safari, Messages, Music, or Notes; instead, Firefox, Signal, Spotify, and Joplin[26].

Yes, I could switch to Android anytime if I wanted. I’ve done in the past, actually.

WWDC 2008 was a pivot moment in my life. 15 years before 2008 it was 1993, and I was finishing high school and then going to spend a few months at the Swiss Army.

---

[19] /blog/iphone-at-the-swiss-apple-store/

[20] /blog/i-want-one/

[21] https://web.archive.org/web/20080728022219/http://www.fuckingnda.com:80/

[22] https://web.archive.org/web/20080829003515/http://twitter.com/akosma/statuses/869619161

[23] https://web.archive.org/web/20080913034550/http://www.ipetitions.com/petition/iPhoneNDA/

[24] /blog/iphone-conference-2008-geneva/

[25] https://vimeo.com/2129015

[26] /blog/joplin/

And in exactly 15 years, in October 1st, 2038, I will retire. My working life will then have spanned 45 years, if I am still alive by 2038, and if the Year 2038 problem[27] doesn't break havoc before, that is.

During my time[28] in the Apple galaxy[29] I made lots and lots of apps[30]. I even brought the first iPads to Switzerland[31] in 2010. I made a living writing Objective-C[32] first, and in comparison, just a little Swift[33] after 2014. A little, only. The reason why I liked making apps was precisely Objective-C. That's why I had bought an iBook G3 in 2002 (a machine that I would later use, interestingly, to install Linux[34] for the first time!) I don't think Swift is a good language for UI development. Dynamic, "late bound" programming languages are much better for that task. Swift is not even good for backend[35] apps[36].

The saddest thing I learnt about Apple was that they do not care at all about the developer experience. They (wrongly) assume it's not their problem. A friend was telling me a few days ago how much of a PITA Xcode has become, getting worse every year, to the point that her fellow Android developers wouldn't even think about making iOS apps, just because of Xcode. And sadly, the only option (albeit limited) to Xcode, JetBrains AppCode, is no longer available.

I learnt that Apple, as far as developers are concerned, is a dead end that requires lots and lots of courage[37].

On the other hand… I've never used as much developer stuff[38] from Microsoft as I have recently. Satya Nadella arguably created a much better ecosystem for software developers than Tim Cook ever will… or want.

---

[27] https://en.wikipedia.org/wiki/Year_2038_problem

[28] /blog/12-years-of-iphone-a-developers-perspective/

[29] /blog/the-developer-guide-to-migrate-across-galaxies/

[30] https://akos.ma/tags/apps/

[31] /blog/first-ipads-in-switzerland/

[32] /tags/objective-c/

[33] /tags/swift/

[34] /blog/ubuntu/

[35] https://akos.ma/blog/fortune-apps/

[36] It feels to me like the industry never learns the lesson. We fall in love with strongly typed languages (Rust these days receiving all the spotlight) and then at some point we get annoyed of the compilation times and start from scratch all over again with less strongly-typed languages.

[37] /blog/courage/

[38] /blog/lots-of-vscode-extensions/


WWDC08
FEATURING
iPHONE
OSX
OSX LEOPARD
JUNE 9-13
MOSCONE
WEST
SAN FRANCISCO

(Credits: Xevio’s t-shirt archive[39])

[39] https://xevio.us/2017/02/t-shirt-archive/#jp-carousel-4732

10 Years of Flat Design

Adrian Kosmaczewski

2023-06-16

On June 10th, 2013, exactly 10 years ago, Apple introduced its non-skeuomorphic, "flat" design idiom, together with iOS 7[1]... and since that moment, for a whole decade, many users have collectively struggled to use their devices properly.

Flat design is a tragedy that doesn't need to be. I have written an article about the subject in De Programmatica Ipsum, so I'll just leave this here.

> As my father approaches his 80th birthday, his first contacts with the latest technologies become ever more interesting. As a pen-and-pencil architect and entrepreneur with 50 years of experience, technology is not a necessity; rather, just another useful tool. He setup his first fax machine at the end of the 90s, and got his first computer a decade after that. At some point in the 2010s he bought a smartphone, and one day asked me for help to finish some task on it. I remember his face when I told him to "touch the button that says 'done'" and he replied, "which button? I do not see any button".
>
> In Spanish, just like in English, the word "button" ("botón") can be used as both "push button" or "shirt button". Being the son of an immigrant costume taylor who moved from Poland to Buenos Aires in the 1930s, my father's brain first considered the latter meaning of the word "button", before the former kicked in.
>
> Truth is, on his smartphone, neither of these things were visible. Nor a push button, nor a shirt button, not a light switch, not a calculator button, not a door knob, not a push lever, nothing. What was visible was, instead, the blue word "Done" eerily floating on the top right corner of the screen.
>
> As my father said, there was no button.

The Button And The Spoon[2] at De Programmatica Ipsum, January 2022.

[1] https://en.wikipedia.org/wiki/IOS_7

[2] https://deprogrammaticaipsum.com/the-button-and-the-spoon/

Still rooting for an end to this madness, and to have contrast again on our screens. Ten years is enough.

Update, 2023-10-27: Just found this old screenshot of Mobile Safari circa 2011…


Carrier
11:26 AM
akos.ma - URL shortener by akosma software
akos.ma/
Search
CORTITO ON GITHUB
CORTITO
AKOS.MA
http://
Key (optional)

Restic

Adrian Kosmaczewski

2023-06-23

One of the greatest discoveries I've made after switching to the Linux galaxy is, without any doubt, the fantastic Restic[1]; a backup tool that deserves to be better known and more widely used.

I usually describe Restic to my friends still in the Apple galaxy[2] as follows:

> It's like Time Machine[3], but on the command line.

Of course, all reductions are dangerous, but the phrase above captures the gist of it.

To install, it's just like any other tool built with Go[4]: download the latest release[5], copy it into your PATH, and you're done. Binaries built with Go are usually self-contained, another of the greatest features of this wonderful programming language.

Well, technically you don't even need to do the above; you can also use it as a container, with Docker or Podman:

```
$ podman pull restic/restic
```

Once you have the program in your path or on your local container image registry, you must create a repository, and for that you need a location (either online, on a shared disk, or on your NAS) for Restic to save backups into.

Once that's done, you can start backing things up:

```
$ restic --repo /somewhere/repo init
$ restic --repo /somewhere/repo backup ~/Documents
```

After that, the `snapshots` subcommand will show the history of your backups:

```
$ restic --repository /somewhere/repo snapshots
```

And as expected, the `restic restore` command... well, restores a particular snapshot into any path you specify.

[1] https://restic.net/
[2] /blog/the-developer-guide-to-migrate-across-galaxies/
[3] https://en.wikipedia.org/wiki/Time_Machine_(macOS)
[4] https://go.dev/
[5] https://github.com/restic/restic/releases

(There's also an `-r` argument instead of `--repo`, of course.)

You can store your repositories in a variety of locations: in a hard disk next to your computer, but also off-site: on an S3 bucket, for example, and in many other locations[6].

Restic is compatible with Windows, Mac, Linux, and even BSD. Your backups are encrypted using your password (which means that losing it will be a catastrophic event) and only stores the changes between backups, making it really efficient. Oh, and it's open-source[7], it can self-update, and even provide autocompletion information for various shells.

Seriously.

I discovered Restic at VSHN[8], because my colleagues used it to create K8up[9], a Kubernetes operator (also written in Go) that helps DevOps engineers backup the contents of their clusters. K8up effectively "wraps" Restic and uses it to drive the backup process; backup repositories created by K8up are actually Restic repositories.

I have been backing up my computers (Mac, Windows, and Linux) with Restic for the past 4 years, and seriously, I'm not switching away from it anytime soon.

[6] https://restic.readthedocs.io/en/stable/030_preparing_a_new_repo.html
[7] https://github.com/restic/restic
[8] https://www.vshn.ch/
[9] https://k8up.io/

Mobile Application Testing Book

Adrian Kosmaczewski

2023-06-30

Going through my archives I found a booklet I wrote 10 years ago, about testing iOS and Android mobile apps. I'm adding it to my Books[1] page for the sake of memory, even though its contents are not at all relevant by today's standards.

I gave printed and electronic versions of this booklet to my students attending a training course with the same name, one that I taught a few times in Switzerland and South Africa.

Here's the link to the PDF[2] and HTML[3] versions, generated with the original implementation of Asciidoc[4] (written in Python, years before I discovered Asciidoctor[5] and its awesomeness) and DocBook. The file is dated "January 2013" but I remember that I gave these trainings until the following year.

The 130 pages booklet contains explanations about how to use several testing techniques for iOS and Android applications from a decade ago, with names that will surely ring a bell to those who were around during those pre-Swift and pre-Kotlin days, when only Objective-C and Java were the languages you used to make mobile apps:

- OCUnit & SenTest
- Kiwi
- QuincyKit
- Frank
- Calabash
- The Android Monkey
- Robolectric
- And the still maintained, awesome, fundamental NSLogger[6] by my friend Florent Pillet[7]!

Ah, le sigh.

[1] /books
[2] /books/Mobile_Application_Testing/Mobile_Application_Testing.pdf
[3] /books/Mobile_Application_Testing/Mobile_Application_Testing.html
[4] https://github.com/asciidoc-py/asciidoc-py
[5] https://asciidoctor.org/
[6] https://github.com/fpillet/NSLogger
[7] https://www.linkedin.com/in/fpillet/

There are also some practical tips and tricks for error handling, coding guidelines, and more. All in all, it was about helping people write better apps, more maintainable, with better standards, and with higher maintainability. Enjoy!

Java Applets in 2023

Adrian Kosmaczewski

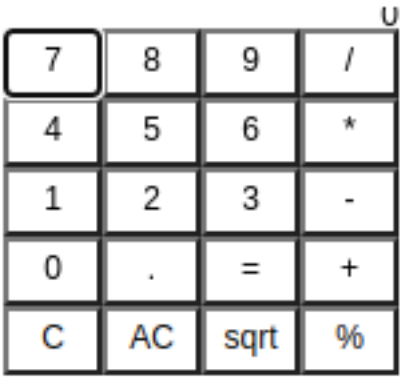
2023-07-07

I've explained recently[1] how to display Macromedia Flash movies in 2023, using Ruffle[2]; today we're going to learn how to run Java Applets in your modern browser of 2023, without having to install Java. Yes, it's possible.

Enter CheerpJ[3]. It's an extension for Google Chrome or Microsoft Edge that somehow transpiles `*.class` files into WebAssembly... which allows a good old applet I wrote in 1997 (!) to render correctly on a "modern" browser of 2023.

To run the calculator, install the extension and then click on it to activate and load the applet. It will appear in the white space between this paragraph and the next. Pretty promise.

This is how the calculator above should look like, in case you're using Firefox or don't have the extension installed:



I wrote it on Windows 3.1[4] (yes, it was still my OS in 1997) using the

[1] /blog/macromedia-flash/
[2] https://ruffle.rs/
[3] https://leaningtech.com/cheerpj-applet-runner/
[4] https://www.howtogeek.com/795478/windows-31-30-years-later/

first 16-bit implementation of the JDK and JRE ever produced: the IBM Applet Development Kit, released August 1996. Don't believe me? Here's an InfoWorld article[5] proving it. A few months later Microsoft released its own[6] 16-bit JVM for Windows 3.1, still a very popular operating system in 1996 and 1997. But I used IBM's.

I don't remember what editor I used to write the code, though; most probably Notepad.

Feel free to download the archive[7] with the complete project, with files dated September 17th, 1997 (hint: it was a Wednesday.) Find below the (admittedly terrible) Java code I wrote in 1997 to make this contraption. My apologies. You've been warned.

```
import java.applet.*;
import java.awt.*;

/** Calculator Applet
    @author Adrian Kosmaczewski, (c) 1997
    @version 1.0
*/

public class CalcApplet extends Applet
{
    String CalcType = "";
    CalcMachine calculator = null;
    Label errorLabel = new Label("Missing <PARAM NAME=\"CalcType\" VALUE=\"Star
    private double firstValue, secondValue;
    private char operationChosen = '=';
    private boolean dotNotAlreadyUsed = true;

    public void init()
    {
        CalcType = getParameter("CalcType");
        if(CalcType == null)     // if the HTML page is wrong...
        {
            add(errorLabel);
        }
        else if(CalcType.equals("Standard") || CalcType.equals("Scientific"))
        //  if the HTML page is correct...
        {
            calculator = new CalcMachine(CalcType);
            add(calculator);
        }
    }

    public boolean action(Event evt, Object obj)  //Modelo 1.0.2 de Java
    {
```

[5] https://www.infoworld.com/article/2077259/ibm-brings-java-to-windows-3-1.html
[6] https://www.cnet.com/tech/tech-industry/windows-3-1-gets-java/
[7] CalcApplet.zip

```
if(obj.equals("1") || obj.equals("2") || obj.equals("3") ||
obj.equals("4") || obj.equals("5") || obj.equals("6") ||
obj.equals("7") || obj.equals("8") || obj.equals("9") ||
obj.equals("0"))
{
    if(calculator.display.getText().equals("0"))
    {
        calculator.display.setText("");
    }
    calculator.display.setText(calculator.display.getText() + obj.toSt
}
if(obj.equals("."))
{
    if(dotNotAlreadyUsed)
    {
        calculator.display.setText(calculator.display.getText() + obj.
        dotNotAlreadyUsed = false;
    }
}
if(obj.equals("+"))
{
    firstValue = getDoubleValue(calculator.display.getText());
    operationChosen = '+';
}
if(obj.equals("-"))
{
    firstValue = getDoubleValue(calculator.display.getText());
    operationChosen = '-';
}
if(obj.equals("*"))
{
    firstValue = getDoubleValue(calculator.display.getText());
    operationChosen = '*';
}
if(obj.equals("/"))
{
    firstValue = getDoubleValue(calculator.display.getText());
    operationChosen = '/';
}
if(obj.equals("="))
{
    secondValue = getDoubleValue(calculator.display.getText());
    switch(operationChosen)
    {
        case('+'):
            showResult(firstValue + secondValue);
            break;
        case('-'):
            showResult(firstValue - secondValue);
            break;
```

```
                case('*'):
                    showResult(firstValue * secondValue);
                    break;
                case('/'):
                    if(secondValue == 0)
                    {
                        calculator.display.setText("Cannot divide by zero");
                    }
                    else
                    {
                        showResult(firstValue / secondValue);
                    }
                    break;
            }
            operationChosen = '=';
        }
        if(obj.equals("AC"))
        {
            calculator.display.setText("0");
            operationChosen = '=';
        }
        if(obj.equals("C"))
        {
            if(operationChosen == '=')
            {
                calculator.display.setText("0");
                operationChosen = '=';
            }
            else
            {
                calculator.display.setText("0");
            }
        }
        if(obj.equals("sqrt"))
        {
            firstValue = getDoubleValue(calculator.display.getText());
            showResult(Math.sqrt(firstValue));
        }
        return true;
    }

    private double getDoubleValue(String s) // Converts from String to double
    {
        if(isOnlyNumbers(s))
        {
            Double tempDouble = new Double(s);
            calculator.display.setText("0");
            return tempDouble.doubleValue();
        }
        else
```

```
            {
                calculator.display.setText("0");
                operationChosen = '=';
                return 0;
            }
        }

        private boolean isOnlyNumbers(String text)
        {
            for(int i=0; i<text.length(); i++)
            {
                if(text.charAt(i)=='0' || text.charAt(i)=='1' || text.charAt(i)=='2
                    text.charAt(i)=='3' || text.charAt(i)=='4' || text.charAt(i)=='
                    text.charAt(i)=='6' || text.charAt(i)=='7' || text.charAt(i)=='
                    text.charAt(i)=='9' || text.charAt(i)=='.')
                {
                    return true;
                }
            }
            return false;
        }

        private void showResult(double d)
        // Converts from double to String and shows the result on the display
        {
            Double tempDouble = new Double(d);
            calculator.display.setText(tempDouble.toString());
        }
    }

    /*************************************************************************

    class CalcMachine extends Panel
    {
        CalcDisplay display = null;
        CalcKeyboard keyb = null;
        int columns = 20;

        public CalcMachine(String type)
        {
            if(type.equals("Standard"))
            {
                columns = 20;
            }
            else if(type.equals("Scientific"))
            {
                columns = 50;
            }
            initialize(type);
        }
```

```
    public void initialize(String type)
    {
        display = new CalcDisplay();
        keyb =  new CalcKeyboard(type);
        setLayout(new BorderLayout());
        add("North", display);
        add("Center", keyb);
    }
}

/*****************************************************************************

class CalcKeyboard extends Panel
{
    CalcButton one = new CalcButton("1");
    CalcButton two = new CalcButton("2");
    CalcButton three = new CalcButton("3");
    CalcButton four = new CalcButton("4");
    CalcButton five = new CalcButton("5");
    CalcButton six = new CalcButton("6");
    CalcButton seven = new CalcButton("7");
    CalcButton eight = new CalcButton("8");
    CalcButton nine = new CalcButton("9");
    CalcButton zero = new CalcButton("0");
    CalcButton point = new CalcButton(".");
    CalcButton plus = new CalcButton("+");
    CalcButton minus = new CalcButton("-");
    CalcButton times = new CalcButton("*");
    CalcButton division = new CalcButton("/");
    CalcButton equal = new CalcButton("=");
    CalcButton clear = new CalcButton("C");
    CalcButton clearAll = new CalcButton("AC");
    CalcButton sqrt = new CalcButton("sqrt");
    CalcButton percent = new CalcButton("%");

    public CalcKeyboard(String type)
    {
        if(type.equals("Standard"))
        {
            setLayout(new GridLayout(5,4));

            add(seven);
            add(eight);
            add(nine);
            add(division);

            add(four);
            add(five);
            add(six);
```

```
            add(times);

            add(one);
            add(two);
            add(three);
            add(minus);

            add(zero);
            add(point);
            add(equal);
            add(plus);

            add(clear);
            add(clearAll);
            add(sqrt);
            add(percent);
        }
        else
        {
            add(new Label("The Scientific Calculator\nis under construction"));
        }
    }
}

/*****************************************************************************

class CalcDisplay extends Label
{
    public CalcDisplay()
    {
        super("0", RIGHT);
    }
}

/*****************************************************************************

class CalcButton extends Button
{
    public CalcButton(String text)
    {
        super(text);
    }
}
```

Serving in the Swiss Army

Adrian Kosmaczewski

2023-07-14

This week it's been 30 years since I first joined the Swiss Army. Involuntarily, that is. I had just finished my Maturité[1] exams, and had subsequently enrolled as a student of Physics in the University of Geneva.

On the morning of Monday, July 10th, 1993, I took the train and around 4 hours later I arrived at Dübendorf[2], a small town north of Zürich, to join the ranks of the "Flieger- und Fliegerabwehr, Nachrichten und Übermittlung Rekrutenschule Dübendorf 93" (School of Recruits in Intelligence and Transmissions of the Air Force and Air Defense, Dübendorf 1993.) Quite a pompous name, if you ask me.

TL;DR: I hated every single second of this experience. Anyway.

(For context, July 1993's BYTE Magazine[3] cover featured the upcoming Pentium chips, 9600 bps modems, and MS-DOS 6 developers explaining DoubleSpace and MemMaker. Good old times.)

Yeah baby, I was in the intelligence forces of the legendary Swiss Air Force. Woop woop. Not as glamorous as what Tom Scott saw recently[4], but yeah, it was in the Air Force. Nothing like Top Gun[5], either, a film that by the way had been released merely 7 years before and was still in everyone's minds. There were a few civil pilots on my squad, who unfortunately, because of health reasons, had not been accepted as military pilots.

I remember going to a small desolate rotten bar in Dübendorf on our first evening outside the base, and the bartender played "In the Army Now"[6] by Status Quo[7]. No comments.

So yeah, just after a few days reality hit hard, and fast. I was in the Air Force Intelligence troops, but it turns out that there were almost no airplanes involved, and by all means, no intelligence whatsoever. I spent the most useless 4 months of my life dealing in the most inane

[1] https://en.wikipedia.org/wiki/Matura#In_Switzerland
[2] https://en.wikipedia.org/wiki/D%C3%BCbendorf
[3] https://archive.org/details/BYTEVol1808199307PentiumPCs
[4] https://www.youtube.com/watch?v=lPaQInkWV7g
[5] https://www.imdb.com/title/tt0092099/
[6] https://en.wikipedia.org/wiki/In_the_Army_Now_(song)
[7] https://en.wikipedia.org/wiki/Status_Quo_(band)

of institutions, learning how to use an analog transmission device that was already obsolete by the time of the Vietnam War.

And this is without counting on the sheer absurdity of an institution that is a real loss of money and talent for a whole country. Only a rich nation like Switzerland can afford such explosive demonstration of futility as this shit.

Looking backwards, I should have done a bit more efforts to be refused entry in such idiocy, but my mother was convinced that I had to be a military officer in order to have a good career in my civil life.

That's how Switzerland used to roll until the 1990s; if you wanted to be a manager in your civil life, you had to be an officer[8] in the army. Because nobody manages people better than screaming drunk officers in charge of soldiers who have no other choice than being there.

Back in 1993 there was no "civil service" as there is now, so my only options as a young Swiss man were:

1. Doing your military service.
2. Being denied entry for health reasons, in which case… you have to pay the Swiss Military Service Exemption Tax[9] (yes, that's a thing.)
3. Going to jail.

So, I'll take option 1 for me, thankyousomuch.

[8] /blog/eight-steps-to-build-a-better-swiss-software-industry/

[9] https://www.ch.ch/en/safety-and-justice/military-service-and-civilian-service/military-service-exemption-tax/

Oh, but I got an immersion in Swiss culture. That's probably the only positive aspect. I got to learn a lot about how Swiss people think (spoiler alert: many don't.) And I got to taste lots of varieties of delicious Swiss wines and cheeses. This was the best part, by far.

And I got my very own, real Swiss Army knife[10], of course; but did you know that there are two variations of it? The one everybody knows and buys as a souvenir, with red plating, that's the "officer's knife," while soldiers get a smaller one plated in non-painted steel.

And what's the main difference between both, you ask? The officer's knife has a corkscrew, while the soldier's one hasn't. You read right: officer's are statistically more prone to open wine bottles than soldiers. That single fact says a lot about the "leadership" in this rotten institution.

(Trivia: it was not a Victorinox, but a Wenger[11]. Both companies ended up merging in 2005.)

I also learnt how to shoot a weapon, in my case the good old SIG SG 510[12] also known as "Sturmgewehr 57," a 7-kilo automatic rifle capable of shooting up to 10 rounds per second (yes, that's 600 per minute) and with enough testosterone in its alloy to make you feel a real man.

Surprisingly, shooting was the only thing that I found interesting or somewhat amusing; even though I got the option to keep the rifle after I finished my service, I decided to return it promptly as soon as I could. Shooting was fun a few times, but I'd rather do other things with my life, thankyousomuch.

Oh, and I even threw a couple of exercise grenades, with half the usual amount of dynamite as a war grenade. It's quite something. But I was very bad at throwing them, so they stopped asking me to do it; and by bad, I mean dangerously clumsy. I wouldn't throw them properly or far away enough. You see what I mean.

To add insult to injury, the end of my military service overlapped with the beginning of my studies at the university. I asked for a special permission to go to class just the first Monday, to get an idea of the location, the timetables, the usual stuff. Said permission request was promptly denied, because you see, serving your country in a uniform trumps all efforts of bringing value to your country as, I don't know, a scientist. We don't need any of those, just fulfill your duties, learn to shoot, and STFU.

And let's not talk about those goddamn "Refresher courses"[13] (aka "Wiederholungskurse" or "Cours de répétition") always conveniently scheduled to happen in the worst possible moment. No, not that they

[10] https://en.wikipedia.org/wiki/Swiss_Army_knife
[11] https://en.wikipedia.org/wiki/Swiss_Army_knife#Victorinox_and_Wenger
[12] https://en.wikipedia.org/wiki/SIG_SG_510
[13] https://www.vtg.admin.ch/en/mein-militaerdienst/aufgebotsdaten.html

fell before exams or anything; they just happened. Once a year. Every year. To be reminded of the pointlessness of all this again, and again, ad nauseam.

Since that day in July 1993 I've systematically voted in favor of the dismantlement of the Swiss Army in each referendum I could. Sadly none of those initiatives actually yield much results, and we still have to deal with drunk soldiers bothering people in train stations on Thursday evenings.

I've already criticised the Swiss Army in a previous post[14]. No need to say more.

Switzerland needs an army, yes: but a professional one. Not a militia. Make it a paid gig, a job, and as I know Swiss culture, I can assure you that there will be queues long hundreds of meters filled with people waiting to join this institution. The current model is a disaster, even after all those reforms.

Update, 2023-12-29: Thus said Leo Tolstoy in "The Kingdom of God Is Within You"[15]:

> Universal military service is for the government the last degree of violence, which is necessary for the support of the whole structure; and for the subjects it is the extreme limit of the possibility of their obedience. It is that keystone which holds the walls and the extraction of which causes the building to cave in.

[14] /blog/swiss-army/

[15] https://standardebooks.org/ebooks/leo-tolstoy/the-kingdom-of-god-is-within-you/leo-wiener

Conway with the Zig Programming Language

Adrian Kosmaczewski

2023-07-21

As suggested in a previous article[1], this year's candidate of my lifetime programming language learning activity is Zig[2], and I decided to reimplement Conway with it[3].

Yes, Zig, apparently the great nemesis of Rust[4].

I wrote this version of Conway when I was still using Ubuntu, where Zig is only available as a snap package; but since I had uninstalled Snap completely (because seriously) I just downloaded the release archive and decompressed it into `~/.local/bin/zig`, adding it to my PATH. Very simple to install, and no Snap thankyousomuch.

In Fedora 38, thankfully, a `sudo dnf install zig` will give you version 0.9.1 (at the time of this writing.)

For this exercise I used version 0.10.1[5] of Zig, installed manually. This is what the source code looks like:

```
const std = @import("std");
const Coord = @import("coord.zig").Coord;
const World = @import("world.zig").World;

var gpa = std.heap.GeneralPurposeAllocator(.{}){};
const allocator = gpa.allocator();

fn clrscr() void {
    std.debug.print("\x1B[2J\x1B[0;0H", .{});
}

pub fn main() !void {
    var alive = try World.blinker(Coord{ .x = 0, .y = 1 }, &allocator);
    defer alive.deinit();
```

[1] /blog/yup-still-learning-a-new-programming-language-every-year/
[2] https://ziglang.org/
[3] https://gitlab.com/akosma/Conway/-/tree/master/Zig
[4] /blog/first-web-app-in-rust/
[5] https://github.com/ziglang/zig/releases/tag/0.10.1

```
    var world = try World.create(30, alive, &allocator);
    var generation: u16 = 0;

    const stdout_file = std.io.getStdOut().writer();
    var bw = std.io.bufferedWriter(stdout_file);
    const stdout = bw.writer();

    while (true) {
        clrscr();
        generation += 1;
        try world.format(stdout);
        try bw.flush();
        std.debug.print("Generation {d}", .{generation});
        std.os.nanosleep(0, 500000000);
        const new_world = try World.evolve(world);
        world.deinit();
        world = new_world;
    }
    world.deinit();
}
```

Modern

The first thing that strikes about Zig is that it is indeed a very modern[6] programming language:

- Built on top of LLVM.
- There is a `zig` command, used to create projects, build them, test them, etc. It is a full build system, which means there's no need for `cmake` or other similar tools.
- Error management is not done by throwing exceptions (there isn't such a thing) but rather using the `defer` and `errdefer` keywords. This is similar to Swift or Go, and Zig even has a matching `try` keyword. This all means that **errors cannot be ignored.**
- Unit tests embedded with your code, just like Go and D do[7]. They can report memory leaks at execution, which is a **very good idea**; this is like having Valgrind[8] built-in. Excellent idea.
- Short compilation times.
- Full family of memory allocators, for various uses, which makes the language very suitable for embedded programming.
- Type inference.
- Generics, implemented with functions returning `type` values.
- Optional types, unwrapped with the `orelse` keyword.
- No macros, no preprocessor, no hidden control flows such as properties that are actually functions, like in D or C#.
- Runtime or compile-time safety mechanisms can be disabled for

[6] https://deprogrammaticaipsum.com/the-great-rewriting-in-rust/
[7] /blog/d-or-what-go-may-have-been/
[8] https://valgrind.org/

faster compilation and execution, if needed.
- Compiles code into small, tight final binaries.
- It features compile-time reflection.
- Fully interoperable with C… to the point that the Zig compiler can even compile C!
- Cross-compilation available off-the-box for a vast array of architectures, ABIs, and stdlib implementations like musl[9], which would make Zig a great option for wrapping executables in container images based on Alpine, for example.
- Asynchronous programming with the usual `async` and `await` keywords.
- `if` and `switch` statements also work as expressions.
- `enum` can have methods.
- `const` correctness similar to C++, which means that a `const AutoHashMap` does not allow its non-const `put()` method to be called, and so on.
- It uses similar string formatting parameters to those of Rust[10].
- There is a VSCode extension[11] and language server[12] for Zig.

And so much more. The language really feels at home in 2023.

Rough Edges

There are a few things that are, in the humble opinion of this author, a bit of a disappointment.

- Zig has no string type; it just uses arrays of u8, which some find cuestionable[13].
  - Maybe this is the reason why I couldn't find any web API frameworks for Zig. After all, web programming is esentially all about string handling, and this is a point where Rust has a major advantage over Zig, even though strings in Rust are far from being a walk in the park.
  - Zig is not meant for using the heap as a primary storage area, which means that string formatting is very similar to C: instead of returning strings on the heap to callers, one passes an allocated formatter into the function returning `void`; one does not return heap-allocated data. This characteristic makes Zig very appropriate for embedded devices and kernel development, where heap allocation is not possible or desired.
- I found compiler errors to be **not** as readable or understandable at first sight, like those generated by the Rust compiler.
- The current version of the VSCode extension at the time of this writing does not offer good code completion or type hints, which

[9] https://musl.libc.org/
[10] https://github.com/ziglang/zig/issues/1358
[11] https://marketplace.visualstudio.com/items?itemName=ziglang.vscode-zig
[12] https://github.com/zigtools/zls/
[13] https://github.com/ziglang/zig/issues/234

hindered a bit the experience of discovering the language.
- The documentation is very much alpha; after all, the project hasn't reached 1.0 yet. Hopefully this situation will improve over time.
- The whole idea of allocators, arguably one of the most fundamental and interesting features of the language, is not very well explained in the documentation.

Useful Links

If you want to learn Zig, here are some links for you:

- Home page[14]
- GitHub project[15]
- ziglearn.org[16], which is arguably the most useful resource I found online.
- Hashmaps explained[17]
- String handling[18]
- Strings in 5 minutes[19]
- A useful and short Zig Cheatsheet[20]

Overall Impression

Zig is a very nice, approachable, and powerful programming language, delivering on its promise of being "a better C than C". However, I believe Rust is a bit higher-level in the abstraction ladder, and maybe more suitable for situations like web applications, thanks to its higher-level libraries and string manipulation abilities.

In any case, Zig really deserves better documentation and its VSCode extension needs some love. At this point in time, the project appears young and full of promise.

[14] https://ziglang.org/
[15] https://github.com/ziglang/zig
[16] https://ziglearn.org/
[17] https://devlog.hexops.com/2022/zig-hashmaps-explained/
[18] https://nofmal.github.io/zig-with-example/string-handling/
[19] https://www.huy.rocks/everyday/01-04-2022-zig-strings-in-5-minutes
[20] https://ratfactor.com/zig/cheatsheet

First Half of 2023 full of AI and ChatGPT

Adrian Kosmaczewski

2023-07-28

We can all agree that the first half of the year has been co-opted by the rise of generative AI, in particular ChatGPT, being used for anything and anyone in any context, disrupting any kind of market or industry just for the sake of it.

I've been collecting links to the things that have caught my attention the most, and here's a quick summary of stuff that happened roughly from January to July 2023 around AI and GPT things.

Impact on Work

This is where these tools will hurt the most, and the most visible effect these days is the writers and actors strike in Hollywood.

- Microsoft-Owned GitHub Lays Off Entire Engineering Team In India, Impacting 142 Roles[1]
- I lost everything that made me love my job through Midjourney over night.[2]
- I am a bad software developer and this is my life[3]
  - In the comments: "learn the interview questions by heart"…
- Freelance writers losing jobs[4]
- Salaries going down[5]
- Employers demanding ChatGPT experience[6]
- The AI Effect: A New Era in Music and Its Unintended Consequences[7] by Rick Beato
- Google "We Have No Moat, And Neither Does OpenAI"[8]
- Developers don't use Stack Overflow that much anymore[9]

[1] https://in.mashable.com/tech/49751/microsoft-owned-github-lays-off-entire-engineering-team-in-india-impacting-142-roles

[2] https://www.reddit.com/r/blender/comments/121lhfq/i_lost_everything_that_made_me_love_my_job/

[3] https://levelup.gitconnected.com/i-am-a-bad-software-developer-and-this-is-my-life-5c248dc72c2a

[4] https://www.reddit.com/r/freelanceWriters/comments/12ff5mw/it_happened_to_me_today/

[5] https://archive.ph/bAhGW

[6] https://archive.ph/ghKO1

[7] https://www.youtube.com/watch?v=-eAQOhDNLt4

[8] https://www.semianalysis.com/p/google-we-have-no-moat-and-neither

[9] https://www.similarweb.com/blog/insights/ai-news/stack-overflow-chatgpt/

- The Leverage of LLMs for Individuals | TL;DR[10]
- Lawyer using ChatGPT[11]
  - Lawyer cites fake cases created by ChatGPT[12]
    - PDF with affidavit[13]
- Artificial Artificial Artificial Intelligence: Crowd Workers Widely Use Large Language Models for Text Production Tasks[14]
- The workers already replaced by artificial intelligence[15]
- An explosion in software engineers using AI coding tools?[16]
- AI is a fad and programming is dead[17]
- If AI was a hammer[18]
- Reports of an AI drone that 'killed' its operator are pure fiction[19]
- Shopify Employee Breaks NDA To Reveal Firm Quietly Replacing Laid Off Workers With AI[20]
- Netflix offering up to $900k for A.I. job as actors and writers strike[21]

Articles in French

- China's job market turned upside down by artificial intelligence[22]
- A 100% virtual radio station with ChatGPT[23]
- Generative AI boosts productivity and levels the playing field[24]
- Artificial intelligence comes to Swiss tourist offices[25]
- Artificial intelligence triggers first redundancies in the US[26]

[10] https://mazzzystar.github.io/2023/05/10/LLM-for-individual/

[11] https://www.nytimes.com/2023/05/27/nyregion/avianca-airline-lawsuit-chatgpt.html

[12] https://simonwillison.net/2023/May/27/lawyer-chatgpt/

[13] https://storage.courtlistener.com/recap/gov.uscourts.nysd.575368/gov.uscourts.nysd.575368.32.1.pdf

[14] https://arxiv.org/abs/2306.07899

[15] https://www.bbc.co.uk/news/business-65906521

[16] https://newsletter.pragmaticengineer.com/p/an-explosion-in-software-engineers

[17] https://dev.to/aschmelyun/ai-is-a-fad-and-programming-is-dead-180f?lidx=54&wpid=363176

[18] https://axbom.com/hammer-ai/

[19] https://www.newscientist.com/article/2376660-reports-of-an-ai-drone-that-killed-its-operator-are-pure-fiction/

[20] https://thedeepdive.ca/shopify-employee-breaks-nda-to-reveal-firm-quietly-replacing-laid-off-workers-with-ai/

[21] https://www.ktvu.com/news/netflix-offering-up-to-900k-for-a-i-job-as-actors-and-writers-strike

[22] https://www.rts.ch/info/sciences-tech/technologies/14042291-en-chine-le-marche-du-travail-est-bouleverse-par-lintelligence-artificielle.html

[23] https://www.rts.ch/info/sciences-tech/13909896-une-radio-100-virtuelle-avec-chatgpt.html

[24] https://www.ictjournal.ch/etudes/2023-05-03/sassister-dune-ia-generative-augmente-la-productivite-et-nivelle-les-competences

[25] https://www.rts.ch/info/economie/14186432-lintelligence-artificielle-debarque-dans-les-offices-de-tourisme-en-suisse.html

[26] https://www.rts.ch/info/economie/14201759-lintelligence-artificielle-provoque-de-premiers-licenciements-aux-etatsunis.html

One in Spanish

- ChatGPT was asked to write a scientific paper: why it made a fool of itself[27]

Quotes

Quote by Jan Wildeboer on Mastodon[28]:

> After the great "success" of #ShadowIT: Introducing #ShadowAI — where employees will feed tons of highly sensitive and internal data and code to some LLM (Large Language Model) like #ChatGPT in the vague hope of becoming more productive or finally getting that promotion. Without any kind of review or approval. This will get people fired. Le sigh. So, so predictable.

A recent quote from Dare Obasanjo on Mastodon[29]:

> When they said generative AI would change everything, I didn't think that meant by causing so many strikes.

Another quote, this time by Cory Doctorow from his piece Google's AI Hype Circle[30]:

> The entire case for "AI" as a disruptive tool worth trillions of dollars is grounded in the idea that chatbots and image-generators will let bosses fire hundred of thousands or even millions of workers.
>
> That's it.

Impact on Society

I don't think we're ready to handle AI as a society, so brace for impact.

- AiLONE[31] by Scott Galloway
- AI and the paperclip problem[32]
- AI to Aid Democracy[33] by Bruce Schneier
- Bing: "I will not harm you unless you harm me first"[34]

[27] https://www.clarin.com/sociedad/pidieron-chatgpt-escribiera-paper-cientifico-hizo-papelon_0_LcQEJBcOK8.html
[28] https://social.wildeboer.net/@jwildeboer/110294375765078887
[29] https://mas.to/@carnage4life/110533969185758897
[30] https://doctorow.medium.com/googles-ai-hype-circle-6158804d1299
[31] https://www.profgalloway.com/ailone/
[32] https://cepr.org/voxeu/columns/ai-and-paperclip-problem
[33] https://www.schneier.com/blog/archives/2023/04/ai-to-aid-democracy.html
[34] https://simonwillison.net/2023/Feb/15/bing/

Impact on Business

New Business Models

AI is giving birth to countless new business ideas; here are some I found.

- ai sum up [35]
- Navigating the AI landscape: a guide for startups[36] by Adrian Tineo
- Ask Rewind[37]
- Enabling Root Cause Analysis[38]
- 10 Ways AI Can Elevate Your Service And Operations Management[39]
- Platform Engineering[40]
- Control Plane[41]
- Markprompt[42]
- Compilation of open LLMs available[43]
- 8 things to know about LLMs[44]

Another quote by Dare Obasanjo (@carnage4life@mas.to)[45]:

> After some consideration, I expect AI will cause companies to find ways to shrink cost centers. Companies will find ways to shrink departments like HR, legal and even IT.
>
> On the other hand, it'll be a force multiplier in context of core businesses versus a cost cutting tool.

Microsoft into Copilots

Microsoft has trademarked the word "Copilot" apparently, and is going all-in with generative AIs.

- Introducing GitHub Copilot X[46]
- Microsoft Security Copilot[47]

[35] https://aisumup.com/

[36] https://www.linkedin.com/pulse/navigating-ai-landscape-guide-startups-adrian-tineo

[37] https://www.rewind.ai/ask-rewind

[38] https://www.logicmonitor.com/support/alerts/aiops-features-for-alerting/enabling-root-cause-analysis

[39] https://www.forbes.com/sites/louiscolumbus/2021/01/29/10-ways-ai-can-elevate-your-service-and-operations-management/?sh=3e1dc2516086

[40] https://serce.me/posts/26-04-2023-platform-engineering-in-the-era-of-llms

[41] https://controlplane.com/

[42] https://markprompt.com/

[43] https://github.com/eugeneyan/open-llms

[44] https://swizec.com/blog/eight-things-to-know-about-llms/

[45] https://mas.to/@carnage4life/110090958951356404

[46] https://github.com/features/preview/copilot-x

[47] https://www.microsoft.com/en-us/security/business/ai-machine-learning/microsoft-security-copilot

- The Microsoft 365 Copilot AI Event in Less than 3 Minutes[48]
- Bing Image Create[49]
- FauxPilot - an open-source alternative to GitHub Copilot server[50]
- Microsoft could offer private ChatGPT to businesses for “10 times” the normal cost[51]
- TurboPilot[52] self-hosted copilot

Services Detecting Plagiarism

The automated generation of text has caused a huge problem of plagiarism, and there’s AI that helps detect text generated by other AI. All very meta! (Source)[53]

- The Most Trusted AI Detector | ChatGPT Detection Tool[54]
- ZeroGPT - Chat GPT, Open AI and AI text detector Free Tool[55]
- Language Model Copilot and API for Businesses | Sapling[56]
- AI-Based Plagiarism & AI Content Detection | Copyleaks[57]
- AI Writing | AI Tools[58]
- Even OpenAI has its own: New AI classifier for indicating AI-written text[59]

Found on Linkedin

I found this list of links on LinkedIn[60], and it’s impressive.

- AutoGPT : A ChatGPT with Automations[61]: Your new best friend. A personal assistant that does the hard work for you!
- KreadoAI_AIGC Digital Marketing Creation/AIGC□□□□□□□□/kreadoai.com[62]: Your global filmmaker. It crafts videos in many languages with virtual people.
- Guide.AI[63]: Your magic video wand. It conjures up explainer videos in a snap and shares ’em easy-peasy.

[48] https://www.youtube.com/watch?v=hGb9UZ8DyDc
[49] https://www.bing.com/images/create
[50] https://github.com/fauxpilot/fauxpilot
[51] https://arstechnica.com/information-technology/2023/05/report-microsoft-plans-privacy-first-chatgpt-for-businesses-with-secrets-to-keep/
[52] https://github.com/ravenscroftj/turbopilot
[53] https://www.bbc.com/future/article/20230720-how-to-spot-an-ai-cheater-artificial-intelligence-large-language-models
[54] https://gowinston.ai/
[55] https://www.zerogpt.com/
[56] https://sapling.ai/
[57] https://copyleaks.com
[58] https://www.turnitin.com/solutions/ai-writing
[59] https://openai.com/blog/new-ai-classifier-for-indicating-ai-written-text
[60] https://www.linkedin.com/posts/juanjosedelgado_ai-work-activity-7074259187039354880-AvyZ
[61] https://autogpt.thesamur.ai/
[62] https://kreadoai.com/
[63] https://guide-ai.com/

- Orimon.ai - No-Code, Sales Enabling, Generative AI bot builder[64]: Your chat buddy. A sales assistant, chatting away powered by ChatGPT.
- Flair[65]: Your personal stylist. It jazzes up your products with professional photos.
- AI Voice Generator: Versatile Text to Speech Software | Murf AI[66]: Your ventriloquist. It can turn any text into voice, how cool is that?
- AI-Powered Meeting Recorder for Zoom and Google Meet - tl;dv[67]: Your scribe. It records, transcribes, and summarizes your meetings, so you don't have to!
- Lexica[68]: Your graphic novel artist. It transforms texts into mind-blowing images for free.
- Free AI-Generated Podcast Summaries | Podsift[69]: Your podcast whisperer. It dishes out juicy summaries of your favorite podcasts.
- Presentation Software | Basic to Beautiful in Minutes with Beautiful.ai[70]: Your creative muse. It takes simple text and transforms it into stunning presentations.
- Arcwise: AI Copilot for Google Sheets[71]: Your spreadsheet guru. An extension that gets your spreadsheets and answers your questions.
- Chatbot - Sttabot[72]: Your app genie. Turn text into AI apps without a single line of code.
- TravelBuddy.ai – AI-Powered Travel Guide[73]: Your trusty travel guide. Your own AI-powered travel assistant, making travel a breeze.
- Use ChatGPT and Claude in Sheets, Docs and Excel[74]: Your spreadsheet speedster. An extension to make your spreadsheets sing.
- Microsoft Designer - Stunning designs in a flash[75]: Your content wizard. It whips up fantastic posts at lightning speed.
- Leading AI video localization & dubbing tool[76]: Your universal translator. It translates videos into a whopping 60+ different languages.
- Fireflies.ai | AI notetaker to transcribe, summarize, analyze meet-

[64] https://orimon.ai/
[65] https://flair.ai/
[66] https://murf.ai
[67] https://tldv.io
[68] https://lexica.art/
[69] https://www.podsift.com/
[70] https://www.beautiful.ai/
[71] https://arcwise.ai/
[72] https://sttabot.io/
[73] https://travelbuddy.ai/
[74] https://gptforwork.com/
[75] https://designer.microsoft.com/
[76] https://www.rask.ai/

ings[77]: Your note-taker. It automates your meeting notes, making productivity a piece of cake.
- TalkBerry - Your personal language tutor, powered by AI[78]: Your language coach. Train your speech in a new language by having a chinwag with this AI.
- Pictory – Home of AI Video Editing Technology[79]: Your film director. It auto-creates short, punchy videos for your networks.
- Skipit.ai[80]: Your cliff notes writer. It summarizes videos and so much more!

Impact in Computer Security

Imagine AIs used to break into systems. Well, guess what, it's already happening.

- AIs as Computer Hackers[81] by Bruce Schneier
- Pentest[82]
- Addressing the Security Risks of AI[83]
  - Links to Report[84]
- WormGPT: New AI Tool Allows Cybercriminals to Launch Sophisticated Cyber Attacks[85]

ThisGPT, ThatGPT

A sure way to raise venture capital (or at least to get some web traffic) in 2023 is to add the suffix (or prefix) "GPT" to your product.

- The quality of ChatGPT 4 degrading?[86]
- How to Train an AI Chatbot With Custom Knowledge Base Using ChatGPT API[87]
- Manage your Kubernetes clusters with AI: K8sGPT[88]
  - Similar one: robusta-dev/kubernetes-chatgpt-bot[89]
- Somebody hacked into ChatGPT, found upcoming plugins[90]
- Minigpt-4[91]

[77] https://fireflies.ai
[78] https://www.talkberry.ai
[79] https://pictory.ai
[80] https://skipit.ai/
[81] https://www.schneier.com/blog/archives/2023/02/ais-as-computer-hackers.html
[82] https://github.com/GreyDGL/PentestGPT
[83] https://www.lawfareblog.com/addressing-security-risks-ai
[84] https://fsi9-prod.s3.us-west-1.amazonaws.com/s3fs-public/2023-04/adversarial_machine_learning_and_cybersecurity_v7_pdf_1.pdf
[85] https://thehackernews.com/2023/07/wormgpt-new-ai-tool-allows.html?m=1
[86] https://news.ycombinator.com/item?id=36134249
[87] https://beebom.com/how-train-ai-chatbot-custom-knowledge-base-chatgpt-api/
[88] https://k8sgpt.ai/
[89] https://github.com/robusta-dev/kubernetes-chatgpt-bot/
[90] https://www.linkedin.com/posts/eric-vyacheslav-156273169_someone-hacked-into-the-chatgpt-api-and-found-activity-7048594542467407873-Cmo5
[91] https://minigpt-4.github.io/

- ChartGPT[92]
- imartinez/privateGPT: Interact privately with your documents using the power of GPT, 100% privately, no data leaks[93]
- GreyDGL/PentestGPT: A GPT-empowered penetration testing tool[94]
- PromtEngineer/localGPT: Chat with your documents on your local device using GPT models. No data leaves your device and 100% private.[95]
- ZueriGPT[96]
- GPT4All[97]
- BlueDolphinGPT[98]
- ChatGPT plugin for Laravel[99] (another one[100])
- AntonOsika/gpt-engineer: Specify what you want it to build, the AI asks for clarification, and then builds it.[101]
- 0xpayne/gpt-migrate: Easily migrate your codebase from one framework or language to another.[102]

Research

There's so much serious investigation going on in this area, it's hard to keep up.

- Boba AI is coming[103]
- BloombergGPT: A Large Language Model for Finance[104]
- SparseGPT: Massive Language Models Can Be Accurately Pruned in One-Shot[105]
- Applied Machine Learning Days[106]
  - Marcel Salathé (@marcelsalathe@mastodon.social)[107]
  - Highlights[108] showing the positive uses of AI & ML
- OpenAI to discontinue support for the Codex API | Hacker News[109]
- Meet GPT4All: A 7B Parameter Language Model Fine-Tuned from a Curated Set of 400k GPT-Turbo-3.5 Assistant-Style

[92] https://www.chartgpt.dev/
[93] https://github.com/imartinez/privateGPT
[94] https://github.com/GreyDGL/PentestGPT
[95] https://github.com/PromtEngineer/localGPT
[96] https://zuerigpt.chregu.tv/
[97] https://gpt4all.io/index.html
[98] https://dolphingpt.ai/
[99] https://benjamincrozat.com/chatgpt-plugin-laravel?lidx=51&wpid=363176
[100] https://barreto.jp/blog/chatgpt-plugin-with-laravel/?lidx=52&wpid=363176
[101] https://github.com/AntonOsika/gpt-engineer
[102] https://github.com/0xpayne/gpt-migrate
[103] https://martinfowler.com/articles/building-boba.html
[104] https://arxiv.org/abs/2303.17564
[105] https://arxiv.org/abs/2301.00774
[106] https://appliedmldays.org/
[107] https://mastodon.social/@marcelsalathe/110094783066921360
[108] https://appliedmldays.org/highlights
[109] https://news.ycombinator.com/item?id=35242069

Generation[110]
- LLM leaderboard[111]
- Why Does Every AI Cycle Start With Chat?[112]
- Unix philosophy for AI[113]

Free and Open Source

The generation, production, and management of FOSS code will be among the first sectors of activity to be completely transformed.

- SUSE: “AI pair programming must not be used.”[114]
- EU AI Act To Target US Open Source Software[115]
- Open source licenses need to evolve to deal with AI[116]
- Open Source LLMs[117]
- Open Source AI Has a New Champion[118]
- Meta’s Llama 2 is not open source[119]

Risks, Hype, and Hysteria

There’s so many people screaming, it’s hard to think.

- Statement for safe AI[120]
- Pause Giant AI Experiments: An Open Letter - Future of Life Institute[121]
- Do not fall for the hype[122]
- Martha Lane Fox warns against hysteria over AI[123]

Learning

Learn more of what’s going on in this space.

- Wildtools.ai[124]
- Alpha Signal | The newsletter for AI professionals[125]

[110] https://www.marktechpost.com/2023/04/03/meet-gpt4all-a-7b-parameter-language-model-fine-tuned-from-a-curated-set-of-400k-gpt-turbo-3-5-assistant-style-generation/
[111] https://huggingface.co/spaces/HuggingFaceH4/open_llm_leaderboard
[112] https://matt-rickard.com/why
[113] https://matt-rickard.com/unix-philosophy-for-ai
[114] https://opensource.suse.com/legal/policy
[115] https://technomancers.ai/eu-ai-act-to-target-us-open-source-software/
[116] https://www.theregister.com/2023/06/23/open_source_licenses_ai/
[117] https://www.schneier.com/blog/archives/2023/06/open-source-llms.html
[118] https://analyticsindiamag.com/open-source-ai-has-a-new-champion/
[119] https://www.theregister.com/2023/07/21/llama_is_not_open_source/?td=rt-3a
[120] https://www.safe.ai/statement-on-ai-risk
[121] https://futureoflife.org/open-letter/pause-giant-ai-experiments/
[122] https://improvingwetware.com/2023/02/07/chatgpt-do-not-fall-for-the-hype
[123] https://www.bbc.co.uk/news/technology-65162257
[124] https://www.wildtools.ai/
[125] https://alphasignal.ai/

- Course: ChatGPT Prompt Engineering for Developers[126]
- ChatGPT for Web Developers[127] by Maximiliano Firtman
- Simon Willison's Weblog[128]
- Learning is fun again[129]
- What Is ChatGPT Doing … and Why Does It Work?[130]

Humour

Finally, some funny stuff. The images are hilarious; credit to whoever created them.

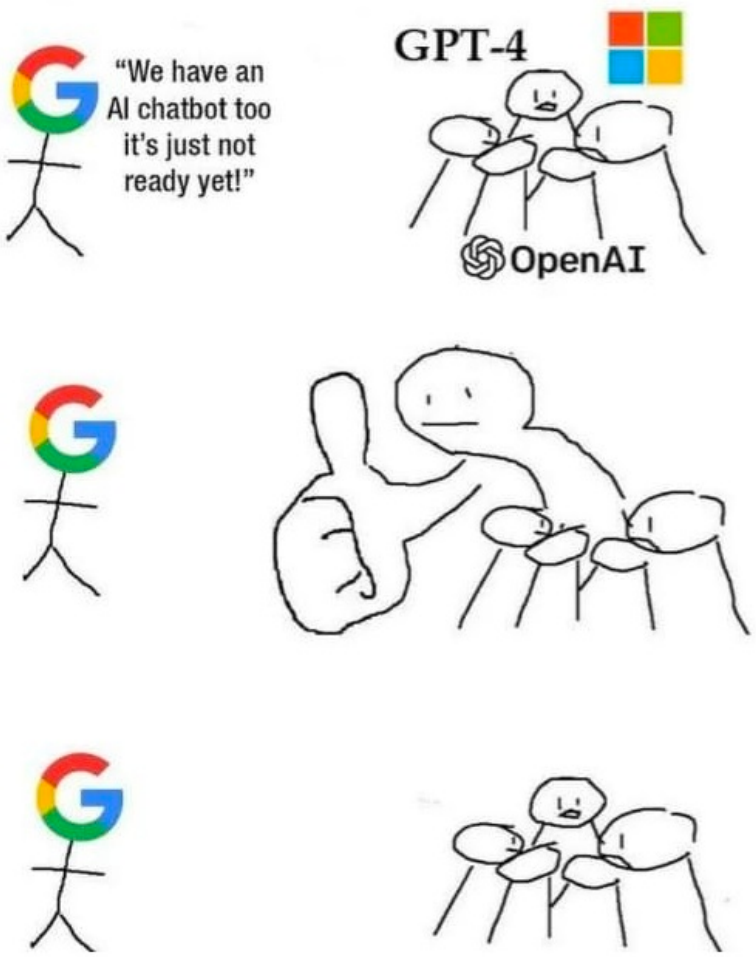
- Companies losing their mind[131]
- ChatShitGPT[132]
- AI founder Anakin Skywalker[133]
- 27[134] (excellent movie!)
- CatGPT[135]
- Dana Fried (@tess@mastodon.social)[136]
- AI Written, AI Read[137] (cartoon by Tom Fishburne)
- PizzaGPT[138] to keep ChatGPT available in Italy
- GPTrillion[139]
- Furby taking over the planet[140]
- Rap between ChatGPT and Google Bard[141]
- DeppGPT[142]
- People are thrilled in ProductHunt[143]
- Guy Who Sucks At Being A Person Sees Huge Potential In AI[144]
- Last digit of π[145]

[126] https://www.deeplearning.ai/short-courses/chatgpt-prompt-engineering-for-developers/
[127] https://firt.dev/chatgpt-web/
[128] https://simonwillison.net/
[129] https://www.vipshek.com/blog/gpt-learning
[130] https://writings.stephenwolfram.com/2023/02/what-is-chatgpt-doing-and-why-does-it-work/
[131] https://www.teamblind.com/post/My-small-no-name-company-has-completely-lost-its-mind-with-AI-nfqEDfSi?utm_source=tldrnewsletter
[132] https://www.chatshitgpt.co.uk/
[133] https://fosstodon.org/@cloudnativeyoda/110312184418093366
[134] https://www.youtube.com/watch?v=dLRLYPiaAoA
[135] https://www.cat-gpt.com/
[136] https://mastodon.social/@tess/110105460869464011
[137] https://marketoonist.com/2023/03/ai-written-ai-read.html
[138] https://www.pizzagpt.it/
[139] https://www.banana.dev/blog/introducing-gptrillion
[140] https://twitter.com/jessicard/status/1642671752319758336
[141] https://www.linkedin.com/posts/danroseman_i-just-got-off-the-waitlist-for-google-bard-activity-7046498751288414209-Ltuv?utm_source=share&utm_medium=member_desktop
[142] https://www.der-postillon.com/2023/05/deppgpt.html
[143] https://mastodon.social/@reichenstein/110492149355397032
[144] https://www.theonion.com/guy-who-sucks-at-being-a-person-sees-huge-potential-in-1850488022
[145] https://aus.social/@haruki_zaemon/110570029910154816

- The Government Needs to Protect us from A.I. | Chad and JT[146]

[146] https://www.youtube.com/watch?v=6WAe0f74Q0s


SD:>_
SwissDevJobs
16,510 followers
27m •
Makes sense, AI will start replacing the less useful positions first
Gaming Company Makes AI Bot Its CEO, Sees Stocks Hit A Record-Breaking High
3
Like
Comment
Repost
Send
Be the first to comment on this



Teachers demostrating against calculators in the 1980s:

AP photo

Elementary school teachers picket against use of calculators in grade school

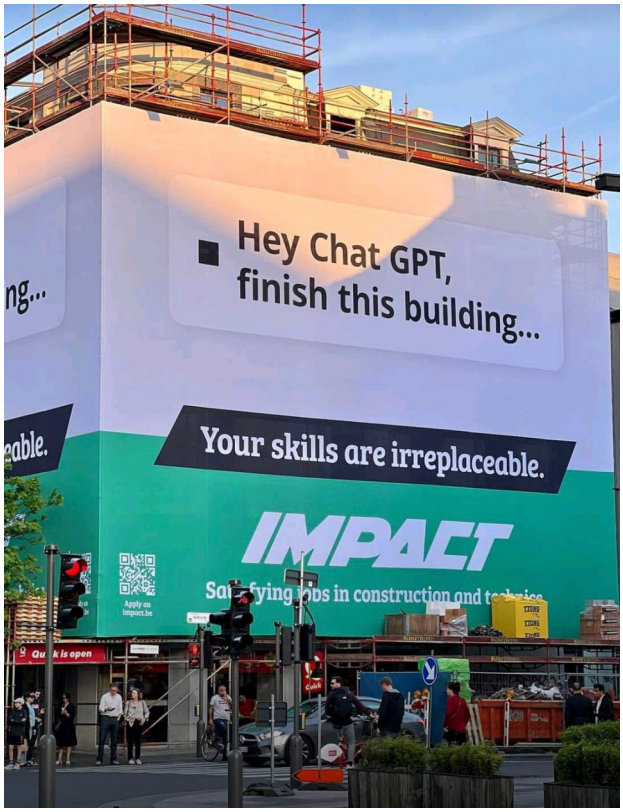
The teachers feel if students use calculators too early, they won't learn math concepts

Math teachers protest against calculator use

By JILL LAWRENCE

"My older kids don't pay any attention to an answer being absurd. strate," he said. "Teachers are shy."


DER SPIEGEL
C 7007 CX
Nr. 16
32. Jahrgang · DM 3,–
17. April 1978
Die Computer-Revolution
Fortschritt macht arbeitslos
DER SPIEGEL
Nr. 36 / 3.9.2016
Deutschland €4,90
Sie sind entlassen!
Wie uns Computer und Roboter die Arbeit wegnehmen – und welche Berufe morgen noch sicher sind
Sportarzt Müller-Wohlfahrt
Heilende Hände, aber Probleme mit den Finanzen
Eine ukrainische Karriere
Erst Gefängnis, dann Präsidentenpalast?
Steinzeit-Drama
Warum Neandertaler zu Kannibalen wurden


Hey Chat GPT, finish this building...
Your skills are irreplaceable.
IMPACT

"The smarter we make the A.I., the less it wants to do our jobs."

(Source)[147]

[147] https://www.instagram.com/p/CuE6yuXoPQz/

Recording Getting Started Guides on Linux

Adrian Kosmaczewski

2023-08-04

As I've said in the past[1], producing videos in Linux isn't very straightforward or stable; it can be a bit of bumpy ride. But I'm stubborn, and Linux is the platform I want to use, so when I was tasked with the creation of "Getting Started" video guides for our products at VSHN[2], I used Linux to create them.

The videos in question are the following, all available at the VSHN YouTube Channel[3], and all around 8 minutes in length:

- Getting Started with Project Syn[4]
- Getting Started with K8up[5]
- Getting Started with APPUiO Cloud[6]
- Getting Started with AppCat[7]

This blog post gives a little bit of insight into the process and tools I used to create these videos.

The Tools

Here are the tools I used for creating these videos[8]:

- OBS Studio[10] and Kooha[11] for screen recording.
- GNOME 42+ includes its own very convenient screen recording[12] feature, but it stores videos in WEBM format.

[1] /blog/video-editing-in-linux/

[2] https://www.vshn.ch/en/

[3] https://vshn.tv/

[4] https://www.youtube.com/watch?v=_6rdYHbly_M

[5] https://www.youtube.com/watch?v=L1xqKtDVGDs

[6] https://www.youtube.com/watch?v=GwP172nGp1g

[7] https://www.youtube.com/watch?v=VgGPlp_KwBs

[8] For information, the Linux distribution I used to create those videos was Ubuntu 22.04. But since May I've been using Fedora 38[9], and so far I'm much happier with it.

[10] https://obsproject.com/

[11] https://github.com/SeaDve/Kooha

[12] Another useful tool you can use for screen recording is Zoom, for example. And if you have a Mac, then QuickTime will get the job done.

- I converted WEBM videos generated by GNOME into MP4 using FFmpeg[13].
- The presentation slides are based in our Asciidoctor-based presentation slides infrastructure[14].
- I plan and rehearse my demos using demo-magic[15].
- I record the audio with Audacity[16].
- I put all the pieces together with Kdenlive[17], my non-linear video editor of choice on Linux. Kdenlive is also cross-platform, so my work can be used and accessed by others in the company (we have people who use Macs and Windows PCs, too, so cross-platform tools are a must.) And it works in computers without a dedicated GPU for rendering; it is slow at doing so, but it gets the job done.

The Process

These are the steps I followed to make those videos:

1. The first and most important thing doesn't have to do with Linux at all, to be honest: I start by **writing a script.** Any video, no matter how short, deserves a nice script. In the cases of these videos I wrote them on our Confluence[18] wiki, so that everyone involved in the team could contribute their ideas. I separate the script into sections, which I use to divide my work in small pieces.
2. Then I use the script to **record the audio** with Audacity. Reading out loud allows me to pace the description of the elements on the screen, to figure out if some screens are missing, and so on. If you want, I end up creating an informal "shooting script"[19] in my head, with images and animations and everything I need to imagine the final product. I create a separate audio file for each section of the script.
3. Then I prepare some **presentation slides** using the asciidoctor-slides[20] infrastructure we use at VSHN for technical presentations. This is based on Asciidoctor+Reveal.js[21] and is text-based, which allows for much faster preparation and rehearsal than any other format. It also works beautifully with code snippets and images.
4. Then I script my **demos** (if any) using Demo Magic, a tool that I've previously talked about[22] in this blog. The result is a script called `demo.sh` that contains all of the steps required to show

[13] https://ffmpeg.org/
[14] https://github.com/vshn/asciidoctor-slides
[15] https://github.com/paxtonhare/demo-magic
[16] https://www.audacityteam.org/
[17] https://kdenlive.org/en/
[18] https://www.atlassian.com/software/confluence
[19] https://nofilmschool.com/shooting-script-example
[20] https://github.com/vshn/asciidoctor-slides
[21] https://docs.asciidoctor.org/reveal.js-converter/latest/
[22] /blog/how-to-use-demo-magic/

some functionality on the terminal.

5. Then, I put on my headphones and hear to my voice speaking while **I record my screen**, clicking on the UI or hitting ENTER on my keyboard, while Demo Magic does its work. Following my own voice makes the final video fit exactly with the audio, which simplifies the editing work on Kdenlive tremendously.
6. Finally I **compose and render** the final video with Kdenlive. I cut some clips that are longer than expected (for example if an operation takes several minutes) and accelerate them, again using Kdenlive. But since I record my screen hearing my voice (see point 5 above) both video and audio clips usually fit perfectly well with one another. Drag, drag, drag, render.
7. When the final project is rendered and the video is uploaded, I **archive the project** using Kdenlive's "archive" feature, creating a nice, self-contained ZIP file with all of the source clips (audio, video, images, etc) and I share that with my team on our shared drive.

To give you an idea, the full production process (from script to upload) of an 8 minutes-long demo requires a whole day of work (around 8 hours.)

More Information

If you're interested in working with video on Linux, here are some articles I wrote about the subject in the past few years you might find interesting:

- How to Use a Microphone[23]
- Live Streaming[24]
- FFmpeg Tips and Tricks[25]
- Video Editing in Linux[26]
- VSHN.timer[27]
- About Remote Conferences[28]

[23] /blog/how-to-use-a-microphone/
[24] /blog/live-streaming/
[25] /blog/ffmpeg-tips-and-tricks/
[26] /blog/video-editing-in-linux/
[27] /blog/vshn.timer/
[28] /blog/about-remote-conferences/

Conway in Minimal BASIC

Adrian Kosmaczewski

2023-08-11

Last Monday I released the 59th issue of De Programmatica Ipsum[1], my dear monthly magazine about code, developers, and society, and this month I talked about BASIC in all of its flavors. As part of the preparation of this issue, I dived into the world of Minimal BASIC code, the one with source code line numbers, the one that would start immediately after powering up your computer, and the one that brings endless nostalgia.

As part of my exploration, I decided to reimplement my venerable Polyglot Conway project[2] in what is usually called "Minimal BASIC", defined as a standard[3] known as ECMA-55[4] and released in 1978.

Writing it

To make it fit in the screen space offered by most emulators, I reduced the size of the grid I used in other versions of the program from 30 by 30 to just 9 by 9, with only a few "objects" animated.

The result of my work, admittedly retro, is right here, with all the bells and whistles you expect from ECMA-55: line numbers, global variables, uppercase statements, `DATA`, `GOTO`, `GOSUB`, the whole package.

In line 100 we initialize the variable `W` (or "World") with a matrix of data read from lines 1000 to 1090. The value 1, as you might expect, represents a location occupied by a living cell, while `0` represents an empty location.

```
100 REM INITIALIZATION
110 LET S = 8
120 LET D$ = "|"
130 DIM W(S, S)
140 DIM Z(S, S)
150 FOR I = 0 TO S
160 FOR J = 0 TO S
```

[1] https://deprogrammaticaipsum.com/

[2] /blog/polyglot-conway/

[3] /blog/basic-standards/

[4] https://www.ecma-international.org/publications-and-standards/standards/ecma-55/

```
170 READ W(I, J)
180 NEXT J
190 NEXT I
200 LET G = 0
```

The variable D$ in line 120 contains a vertical bar, separated on its own, so that we can change its value (for example, in the Commodore 64, we change it to an uppercase B, which in PETSCII is represented as a vertical bar. Don't ask.)

The variable S contains the maximum index of rows or columns in the grid. Since we iterate from zero, the final grid is 9 by 9.

From lines 370 to 400 we print the grid system on top and to the left of the world. In lines 420 and 430 we print an X at each living cell, and a blank space for the rest.

In line 440 we avoid printing a final vertical bar if we're at the end of the line, to save some screen space.

We use two GOSUB statements here (lines 380 and 510), and both "procedures" are described below in this article. Line 520 contains a nice GOTO statement that makes this part of the code repeat ad nauseam.

```
350 REM PRINT GRID IN AN ENDLESS LOOP
360 PRINT ""
370 FOR A = 0 TO S
380 IF A = 0 THEN GOSUB 600
390 PRINT A;
400 PRINT D$;
410 FOR B = 0 TO S
420 IF W(B, A) = 0 THEN PRINT "   ";
430 IF W(B, A) = 1 THEN PRINT " X ";
440 IF B < S THEN PRINT D$;
450 NEXT B
460 PRINT ""
470 NEXT A
480 LET G = G + 1
490 PRINT ""
500 PRINT "GENERATION ", G
510 GOSUB 700
520 GOTO 350
```

In line 380 we GOSUB to line 600 to print the first line of the grid system, and then we return. GOSUBs are the closest thing we have to subroutines in Minimal BASIC.

Also, remember: **all variables are global!**

```
600 REM FIRST LINE WITH COORDINATES
610 FOR B = 0 TO S
620 IF B = 0 THEN PRINT "     0 ";
630 IF B > 0 THEN PRINT B;
```

```
640 IF B < S THEN PRINT D$;
650 NEXT B
660 PRINT ""
670 RETURN
```

From lines 700 to 900 we have a routine that makes the `W` variable evolve to its next state. From 710 to 760 we reset all values in `W` to zero; but don't worry, we make a copy of `W` called `Z` in line 730.

Then from lines 770 to 900 we have a series of nested `FOR` loops. We iterate cell by cell, and for each, we count the number of living organisms around them, storing it in a variable called `C`.

In line 850 we subtract the value of the current cell, because we only want to count the number of living neighbors.

Finally, we apply the evolution algorithm: if the cell was alive (line 860) it stays alive only if it had 2 or 3 neighbors. Otherwise, it dies, either of boredom or of suffocation. Poor thing.

If the cell was dead (line 870) it becomes alive if it has three neighbors (I won't comment on the practical implications of such number.) We set the values in the `W` global variable, and we return to line 520, which is a `GOTO` to line 350, to start the process all over again.

```
700 REM EVOLVE WORLD TO NEXT GENERATION
710 FOR I = 0 TO S
720 FOR J = 0 TO S
730 LET Z(I, J) = W(I, J)
740 LET W(I, J) = 0
750 NEXT J
760 NEXT I
770 FOR X = 0 TO S
780 FOR Y = 0 TO S
790 LET C = 0
800 FOR A = X - 1 TO X + 1
810 FOR B = Y - 1 TO Y + 1
820 IF A >= 0 AND B >= 0 AND A <= S AND B <= S THEN LET C = C + Z(A,B)
830 NEXT B
840 NEXT A
850 LET C = C - Z(X, Y)
860 IF Z(X, Y) = 1 AND (C = 2 OR C = 3) THEN LET W(X, Y) = 1
870 IF Z(X, Y) = 0 AND (C = 3) THEN LET W(X, Y) = 1
880 NEXT Y
890 NEXT X
900 RETURN
```

At the end of the program, we have the `DATA` statements used to initialize the `W` variable on top of the program, and a mandatory `END` statement at the end, even if we never really reach it, thanks to the `GOTO` in line 520.

It's not hard to see the initial configuration of the grid through the ones scattered in the `DATA`.

```
1000 REM INITIALIZATION DATA
1010 DATA 0, 1, 0, 0, 0, 0, 0, 0, 0
1020 DATA 0, 1, 0, 0, 0, 0, 1, 0, 0
1030 DATA 0, 1, 0, 0, 1, 0, 1, 0, 0
1040 DATA 0, 0, 0, 0, 0, 1, 1, 0, 0
1050 DATA 0, 0, 0, 0, 0, 0, 0, 0, 0
1060 DATA 0, 0, 0, 0, 0, 0, 0, 0, 0
1070 DATA 0, 0, 1, 0, 0, 0, 0, 0, 0
1080 DATA 0, 1, 0, 1, 0, 0, 0, 0, 0
1090 DATA 0, 0, 1, 0, 0, 0, 0, 0, 0

9999 END
```

Et voilà :)

Running it

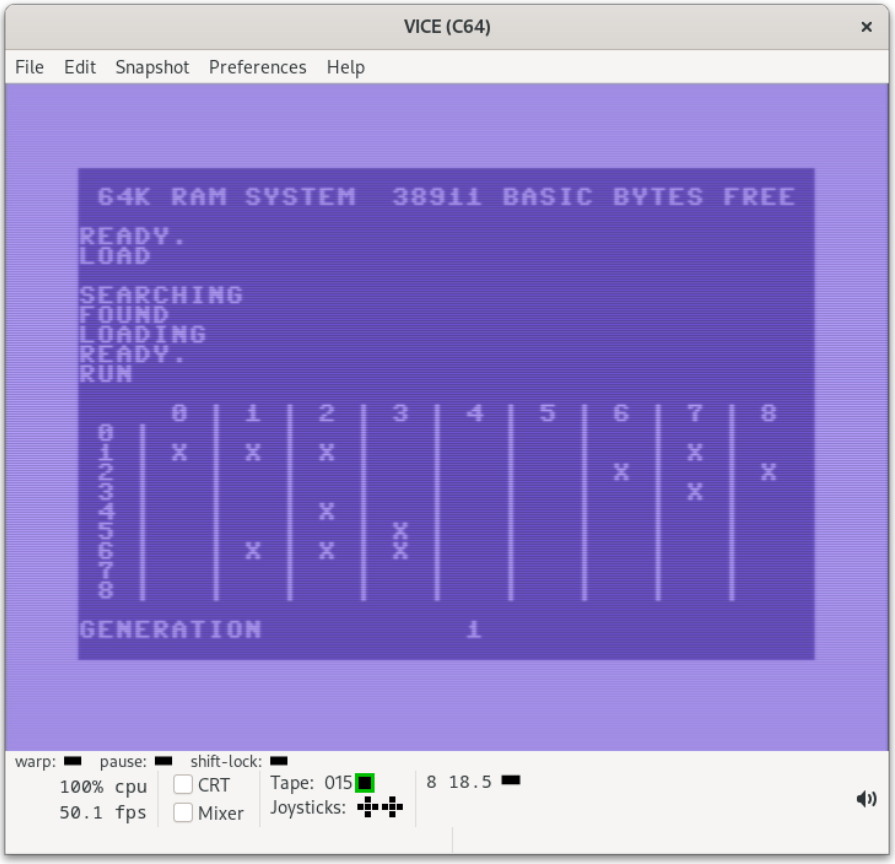
This code runs **verbatim** (and compiles with FreeBASIC) with the following interpreters and compilers, all tested on Fedora 38[5].

- VICE - the Versatile Commodore Emulator[6]
  - Attach the datasette image `conway.tap` (menu "File / Attach datasette image...") and type `LOAD`. Select the `play` command (click on the "Tape" word at the bottom of the emulator and select "Play"), and wait for the program to load. Then type `LIST` to see the code, and then `RUN`. Press `ESC` to stop.
  - You can also paste the code using the "ALT+F9" (to reset the emulator) and "ALT+INSERT" key combinations (to paste the code as if it were entered manually) but then you'll need to manually change line 120 before running, as follows: `120 LET D$ = "B"` (uppercase B in PETSCII[7] is a vertical bar)
  - Only tested on the Commodore 64 emulator of the suite.

[5] /blog/fedora-38/

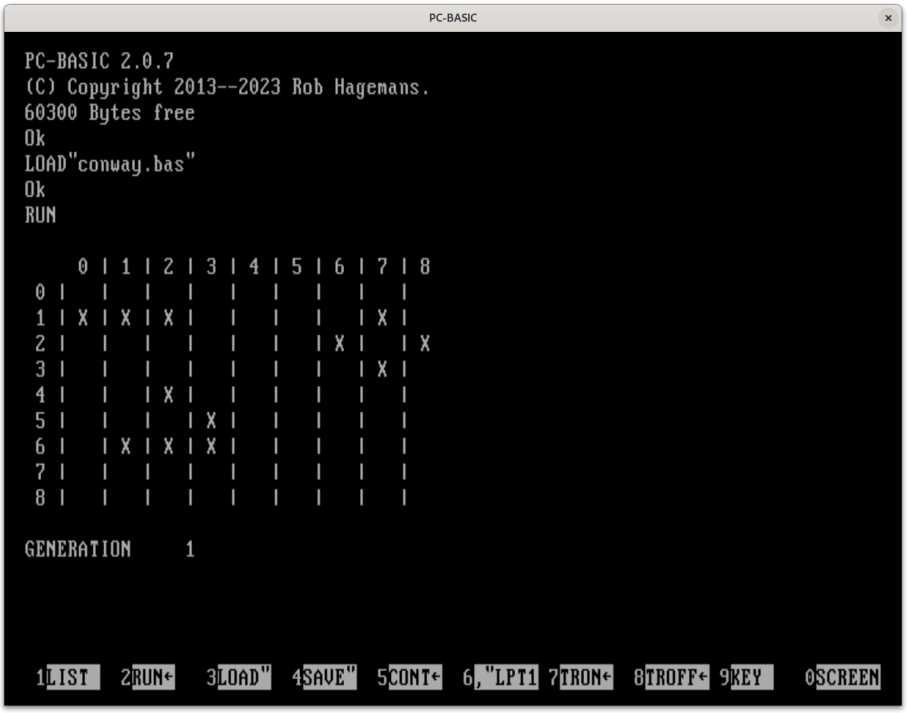
[6] https://vice-emu.sourceforge.io/

[7] https://en.wikipedia.org/wiki/PETSCII



- FreeBASIC 1.10.0[8]
  - Use the `fbc conway.bas -lang qb -x bin/conway` command to compile, and then run `bin/conway`. CTRL+C to stop.
  - You will get an error if you forget the `-lang qb` argument!
  - Tested on FreeDOS[9] and Fedora 38.
- PC-BASIC[10]
  - Run the program with `pcbasic conway.bas`. CTRL+C to stop. Close the window to quit.
  - You can also just launch `pcbasic`, press F3, and load `conway.bas`. Press F1 to `LIST` and F2 to `RUN`. CTRL+C to stop.

[8] https://www.freebasic.net/
[9] https://freedos.org/
[10] https://robhagemans.github.io/pcbasic/



- Bywater BASIC Interpreter[11]
  - Run the program with `bwbasic conway.bas`. CTRL+C to stop.
  - You can also just launch `bwbasic`, type `LOAD`, and enter `conway.bas`. Type `LIST` and `RUN`. CTRL+C to stop. Type `quit` to exit the program. Tested on FreeDOS[12] and Fedora 38.
- QB64[13]
  - Open the `conway.bas` file in the editor and press F5 to compile and run.
- Vintage BASIC[14]
  - Run the program with `vintbas conway.bas`. CTRL+C to stop.
- Chipmunk Basic[15]
  - Run the program with `basic conway.bas`. CTRL+C to stop. Type `quit` to exit the program.
- SmallBASIC[16]
  - Download the AppImage file, and `chmod +x` it. Then run it at the console with `conway.bas` as the only parameter.
  - Not to be confused with Small Basic[17], from Microsoft, which

[11] https://sourceforge.net/projects/bwbasic/
[12] https://freedos.org/
[13] https://qb64.com/
[14] http://www.vintage-basic.net/
[15] https://www.nicholson.com/rhn/basic/
[16] https://smallbasic.github.io/
[17] https://smallbasic-publicwebsite-code.azurewebsites.net/

is not compatible with this project.

- Microsoft QBASIC[18] on MS-DOS 5.0[19] and later.

I've also managed to run the code verbatim on some online emulators:

- Owlet BBC BASIC Editor[20]
  - Clicking on this link[21] will load and start the program directly

[18] https://en.wikipedia.org/wiki/QBasic

[19] https://en.wikipedia.org/wiki/MS-DOS#MS-DOS_5.x

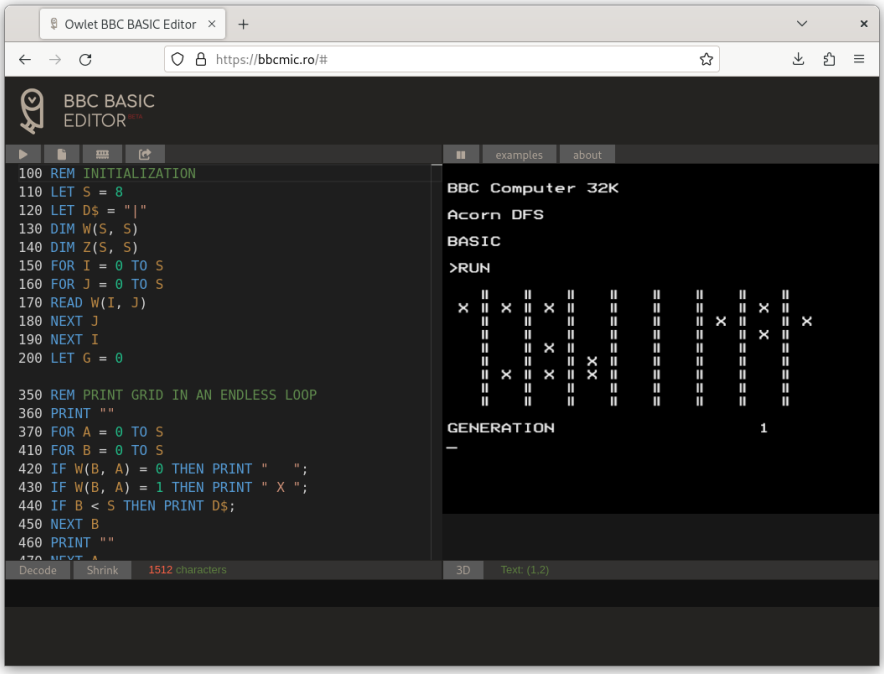
[20] https://bbcmic.ro/

[21] https://bbcmic.ro/#%7B%22v%22%3A1%2C%22program%22%3A%22100%20REM%20INITIALIZATION%5Cn110%20LET%20S%20%3D%208%5Cn120%20LET%20D%24%20%3D%20%5C%22%7C%5C%22%5Cn130%20DIM%20W%28S%2C%20S%29%5Cn140%20DIM%20Z%28S%2C%20S%29%5Cn150%20FOR%20I%20%3D%200%20TO%20S%5Cn160%20FOR%20J%20%3D%200%20TO%20S%5Cn170%20READ%20W%28I%2C%20J%29%5Cn180%20NEXT%20J%5Cn190%20NEXT%20I%5Cn200%20LET%20G%20%3D%200%5Cn%5Cn350%20REM%20PRINT%20GRID%20IN%20AN%20ENDLESS%20LOOP%5Cn360%20PRINT%20%5C%22%5C%22%5Cn370%20FOR%20A%20%3D%200%20TO%20S%5Cn410%20FOR%20B%20%3D%200%20TO%20S%5Cn420%20IF%20W%28B%2C%20A%29%20%3D%200%20THEN%20PRINT%20%5C%22%20%20%20%5C%22%3B%5Cn430%20IF%20W%28B%2C%20A%29%20%3D%201%20THEN%20PRINT%20%5C%22%20X%20%5C%22%3B%5Cn440%20IF%20B%20%3C%20S%20THEN%20PRINT%20D%24%3B%5Cn450%20NEXT%20B%5Cn460%20PRINT%20%5C%22%5C%22%5Cn470%20NEXT%20A%5Cn480%20LET%20G%20%3D%20G%20%2B%201%5Cn490%20PRINT%20%5C%22%5C%22%5Cn500%20PRINT%20%5C%22GENERATION%20%5C%22%2C%20G%5Cn510%20GOSUB%20700%5Cn520%20GOTO%20350%5Cn%5Cn600%20REM%20FIRST%20LINE%20WITH%20COORDINATES%5Cn610%20FOR%20B%20%3D%200%20TO%20S%5Cn620%20IF%20B%20%3D%200%20THEN%20PRINT%20%5C%22%20%20%20%20%200%20%5C%22%3B%5Cn630%20IF%20B%20%3E%200%20THEN%20PRINT%20B%3B%5Cn640%20IF%20B%20%3C%20S%20THEN%20PRINT%20D%24%3B%5Cn650%20NEXT%20B%5Cn660%20PRINT%20%5C%22%5C%22%5Cn670%20RETURN%5Cn%5Cn700%20REM%20EVOLVE%20WORLD%20TO%20NEXT%20GENERATION%5Cn710%20FOR%20I%20%3D%200%20TO%20S%5Cn720%20FOR%20J%20%3D%200%20TO%20S%5Cn730%20LET%20Z%28I%2C%20J%29%20%3D%20W%28I%2C%20J%29%5Cn740%20LET%20W%28I%2C%20J%29%20%3D%200%5Cn750%20NEXT%20J%5Cn760%20NEXT%20I%5Cn770%20FOR%20X%20%3D%200%20TO%20S%5Cn780%20FOR%20Y%20%3D%200%20TO%20S%5Cn790%20LET%20C%20%3D%200%5Cn800%20FOR%20A%20%3D%20X%20-%201%20TO%20X%20%2B%201%5Cn810%20FOR%20B%20%3D%20Y%20-%201%20TO%20Y%20%2B%201%5Cn820%20IF%20A%20%3E%3D%200%20AND%20B%20%3E%3D%200%20AND%20A%20%3C%3D%20S%20AND%20B%20%3C%3D%20S%20THEN%20LET%20C%20%3D%20C%20%2B%20Z%28A%2CB%29%5Cn830%20NEXT%20B%5Cn840%20NEXT%20A%5Cn850%20LET%20C%20%3D%20C%20-%20Z%28X%2C%20Y%29%5Cn860%20IF%20Z%28X%2C%20Y%29%20%3D%201%20AND%20%28C%20%3D%202%20OR%20C%20%3D%203%29%20THEN%20LET%20W%28X%2C%20Y%29%20%3D%201%5Cn870%20IF%20Z%28X%2C%20Y%29%20%3D%200%20AND%20%28C%20%3D%203%29%20THEN%20LET%20W%28X%2C%20Y%29%20%3D%201%5Cn880%20NEXT%20Y%5Cn890%20NEXT%20X%5Cn900%20RETURN%5Cn%5Cn1000%20REM%20INITIALIZATION%20DATA%5Cn1010%20DATA%200%2C%201%2C%200%2C%200%2C%200%2C%200%2C%200%2C%200%5Cn1020%20DATA%200%2C%201%2C%200%2C%200%2C%200%2C%200%2C%201%2C%200%2C%200%5Cn1030%20DATA%200%2C%201%2C%200%2C%200%2C%201%2C%200%2C%200%2C%200%2C%200%2C%200%2C%201%2C%201%2C%200%2C%200%5Cn1050%20DATA%200%2C%200%2C%200%2C%200%2C%200%2C%200%2C%200%2C%200%2C%200%5Cn1060%20DATA%200%2C%200%2C%200%2C%200%2C%200%2C%200%2C%200%2C%200%2C%200%5Cn1070%20DATA%200%2C%200%2C%201%2C%200%2C%200%2C%200%2C%200%2C%200%5Cn1080

on the emulator.

- Otherwise, paste the code of `conway.bas` on the editor and press the “Play” button. To make the output more readable, remove the following lines used to print the coordinate system above and to the left of the grid:

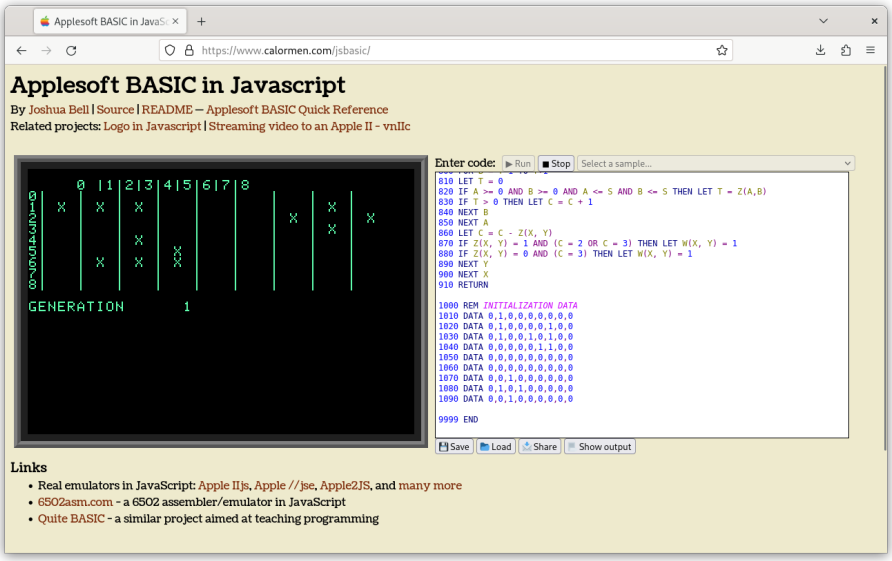
```
380 IF A = 0 THEN GOSUB 600
390 PRINT A;
400 PRINT D$;
```



- Online AppleSoft BASIC in JavaScript[22]
  - Paste the code of `conway.bas` on the editor and press “Run”.

%20DATA%200%2C%201%2C%200%2C%201%2C%200%2C%200%2C%200%2C%200%2C%200%5Cn1090%20DATA%200%2C%200%2C%201%2C%200%2C%200%2C%200%2C%200%2C%200%2C%200%5Cn%5Cn9999%20END%5Cn%22%7D

[22] https://www.calormen.com/jsbasic/



- Decimal BASIC[23]
  - Select the “Option / Syntax” menu and select “Obsolete Minimal BASIC”. Paste the code on the editor and hit the play button.
- 8bitworkshop IDE[24]
  - Paste the code of `conway.bas` on the editor and press the “Play” button.
- Quite BASIC[25]
  - It almost works off-the-box: you must change the comma for a + sign on line 500 before running, because Quite BASIC does not recognize commas on `PRINT` statements.

It was a fun experience to discover the incredible community of users online that keep the memory of BASIC alive, providing emulators of all kinds, both online and offline. I was pleased to see the fervor and the love people put in these things, and it was a fantastic trip 40 years in the past.

More Conway in BASIC

With this implementation, there are now ~~5~~ 6 different dialects of BASIC in the Conway project:

- Minimal BASIC[26], described in this page.
- QBasic[27], using Sub procedures instead of `GOSUB`, and getting rid of the `GOTO` statement with a nicer `Do… Loop`, but still with `DATA`

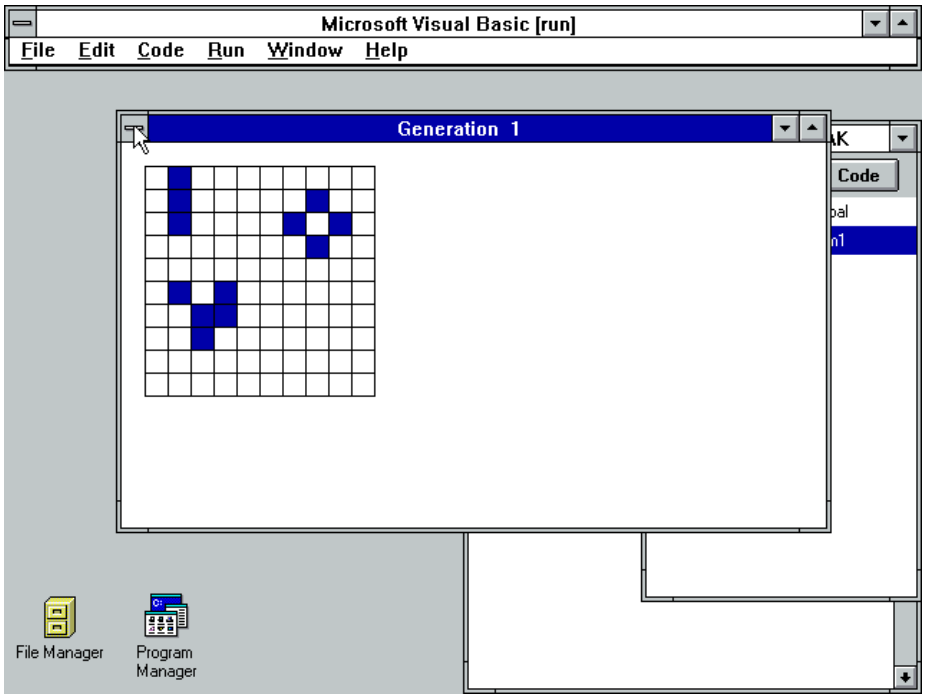
[23] http://hp.vector.co.jp/authors/VA008683/english/
[24] https://8bitworkshop.com/v3.10.1/?platform=basic&file=hello.bas
[25] https://www.quitebasic.com/
[26] https://gitlab.com/akosma/Conway/-/tree/master/MinimalBASIC
[27] https://gitlab.com/akosma/Conway/-/tree/master/QBasic

statements.

- VBScript[28] to be run on Windows 11 using the `cscript` command.
- FreeBASIC[29] using include files and other features of this compiler.
- VB.NET[30], probably the most “modern” version of all.

The last three ones are those that look more similar to other projects. In the first three cases, I had to adapt the code to the various constraints of those BASIC dialects.

Update, 2023-08-18: Added a sixth BASIC dialect with a version for Visual Basic 1.0[31], thanks to WinWorld[32] offering downloads of MS-DOS 5.0[33], Windows 3.1[34], and Visual Basic 1.0[35], ready to be conveniently installed and run in a VirtualBox VM[36].



[28] https://gitlab.com/akosma/Conway/-/tree/master/VBScript
[29] https://gitlab.com/akosma/Conway/-/tree/master/FreeBASIC/
[30] https://gitlab.com/akosma/Conway/-/tree/master/VB.NET
[31] https://gitlab.com/akosma/Conway/-/tree/master/VisualBasic
[32] https://winworldpc.com/
[33] https://winworldpc.com/product/ms-dos/50
[34] https://winworldpc.com/product/windows-3/31
[35] https://winworldpc.com/product/microsoft-visual-bas/10
[36] https://www.virtualbox.org/

Kosmaczewski's Law

Adrian Kosmaczewski

2023-08-18

I was talking recently with Graham[1] and an observation came to my mind, one that I hereby state as Kosmaczewski's Law: "The flow of knowledge is inverse to the flow of fascism".

I'm sure many others have said the same or similar thing in one way or another, but I always wanted to coin a law with my family name[2]. Graham mentioned something similar on our conversation:

> heh, I have heard the corollary stated before: "reality tends to be left wing"

which is an adaptation of something Stephen Colbert said[3]:

> Reality has a well-known liberal bias.

Let's look for some examples:

- The incredibly large number of scientists that fled Nazism from Europe towards the USA before World War II, including Kurt Gödel, Albert Einstein, John von Neumann[4], John Kemeny[5], and all the other Martians[6].
- La Noche de los Bastones Largos[7] of 1966 which caused the most drastic brain drain ever seen in Argentina's history, and one which the country has never fully recovered from.
- Thanks to Melon's new policies on X-tter, thousands of scientists are moving out of Twitter[8] and towards Mastodon.
- Canada is poaching tech workers with a shovel[9] as America prioritizes religion over science.

[1] https://www.sicpers.info/

[2] I feel like my family name goes well with such things like scientific laws, don't you think?

[3] https://www.psychologytoday.com/us/blog/inconvenient-facts/201905/do-conservatives-or-liberals-hold-more-biased-perceptions

[4] https://deprogrammaticaipsum.com/william-aspray/

[5] https://deprogrammaticaipsum.com/dartmouth-college/

[6] https://en.wikipedia.org/wiki/The_Martians_(scientists)

[7] https://en.wikipedia.org/wiki/La_Noche_de_los_Bastones_Largos

[8] https://www.nature.com/articles/d41586-023-02554-0

[9] https://www.thestar.com/business/how-canada-poached-10-000-tech-workers-from-the-u-s-in-just-48-hours/article_c159c7cc-6163-5414-8453-0db70899df90.html

In my little head, this law is what explains that, even if the BRICS[10] block might surpass the GDP of G7 countries soon, they will **not** develop into healthy, stable, long-running basis for new knowledge and science. Instead, they will rather rot into populism, misinformation, religious fanatism, and prosecution of free thinking. And worse.

Each and every BRICS country is a perfect example of how not to attract and retain brains. They are, however, excellent examples of demagogy, tyranny, oppression, misogyny, fundamentalism, exhacerbation of nationalism, prosecution of opposition, press censorship, cult of the motherland, and anything that can become the antithesis of a healthy, thinking society.

Just look at the top picture on the BRICS Wikipedia article[11] and think for a minute. And yes, I write these words precisely as they prepare for their summit in South Africa[12].

Rather, it's the positive, inward flow of healthy, thinking brains, attracted by freedom of thought and respect for minorities of all kinds, which determines the wealth of a nation in the long run. Nations diving into populism and fundamentalism will recess into an abyss of ignorance and ignominy; because thinking heads will pack up and leave them as soon as they can.

Following the Canadian example I mentioned previously, I think that there is an opportunity for some countries in the world to take a giant leap forward nowadays, by poaching talent from other cultures decaying because of fascists tendencies. I've mentioned this in my article about John von Neumann at De Programmatica Ipsum[13] in July 2022:

> Where will the von Neumanns, the Einsteins, and the Gödels of our age migrate after the SCOTUS overrules democracy[14] in 2024 and Gödel's Loophole[15] is proven to exist? At this time, the two major powers of our time, the United States and China seem unlikely locations for brains to settle and grow in the long term. Maybe this is the chance of a lifetime for the European continent; if not for all, at least for some countries therein: Portugal, Scandinavia, The Netherlands, and Switzerland, for example. Overseas, New Zealand, Israel, Japan, South Korea, and Canada also seem like potential candidates to become major research powers in the near future.

[10] https://en.wikipedia.org/wiki/BRICS

[11] https://en.wikipedia.org/wiki/BRICS

[12] https://www.reuters.com/world/key-facts-about-brics-2023-summit-2023-08-07/

[13] https://deprogrammaticaipsum.com/william-aspray/

[14] https://twitter.com/thom_hartmann/status/1543079225254559744

[15] https://en.wikipedia.org/wiki/G%C3%B6del\XeTeXglyph\numexpr\XeTeXcharglyph ”0027\relax{}s_Loophole

(I wrote this in 2022, but recent[16] events[17] make me think that it was a mistake to add Israel to the list.)

For those who read "Brave New World"[18] by Aldous Huxley, that place would be Iceland. Sorry for the spoiler to those who haven't read it.

So far, Switzerland could be doing better in this respect, but here we have our own fascist, far-right political party (very fond of those BRICS countries and their policies, by the way) to deal with first. Not even Switzerland is immune from the leprosy of such political movements.

Of course, Kosmaczewski's Law is not the only way to explain the long-term riches of a nation, nor it has any scientific basis, as it is based on personal reflection, myopia, and inherent ignorance of the author of these lines. Ray Dalio[19] offers an admittedly much more researched and thorough perspective, based on economic flows and military power, in his book "The Changing World Order"[20], which I strongly recommend everyone to read.

I think Kosmaczewski's Law also applies in the great cycles of history mentioned by Mr. Dalio. At some point, nations start rotting inside, they clutch at the straws of fundamentalism and religion, start chasing out scientists, and a new generation goes somewhere else where they can keep on creating new science, away from prosecution and religious nonsense.

So, there you go, Kosmaczewski's Law: **"The flow of knowledge is inverse to the flow of fascism."** You can quote it if you want. And if someone said this first, or better, just quote them instead. As a TED conference would say, this is an idea worth spreading.

Update, 2023-08-25: Here's Kosmaczewski's Law in action[21] (thanks Graham for the link!)

[16] https://www.msn.com/en-in/news/other/israeli-tech-startups-found-to-flock-to-the-us-heres-why/ar-AA1fkLr3

[17] https://www.jta.org/2023/03/20/israel/alarmed-by-their-countrys-political-direction-more-israelis-are-seeking-to-move-abroad

[18] https://en.wikipedia.org/wiki/Brave_New_World

[19] https://en.wikipedia.org/wiki/Ray_Dalio

[20] https://en.wikipedia.org/wiki/The_Changing_World_Order

[21] https://www.theregister.com/2023/08/18/chatgpt_political_bias/

Conway in Rexx, Cobol, and Fortran

Adrian Kosmaczewski

2023-08-25

Here's more dabbling in programming languages to re-create my venerable interpretation of Conway's Game of Life[1], this time using three stereotypical languages of the IBM galaxy[2]: the Rexx scripting language, good old COBOL, and Fortran 95.

Rexx

You probably never heard of Rexx before; that's all right, here's what Wikipedia[3] has to say about it:

> Rexx (Restructured Extended Executor) is a programming language that can be interpreted or compiled. It was developed at IBM by Mike Cowlishaw[4]. It is a structured, high-level programming language designed for ease of learning and reading. Proprietary and open source Rexx interpreters exist for a wide range of computing platforms; compilers exist for IBM mainframe computers.

What can you use it for?

> Rexx is a full language that can be used as a scripting, macro language, and application development language. It is often used for processing data and text and generating reports; this means that Rexx works well in Common Gateway Interface (CGI) programming and is used for this purpose, like later languages such as Perl. Rexx is the primary scripting language in some operating systems, e.g. OS/2, MVS, VM, AmigaOS, and is also used as an internal macro language in some other software, such as SPF/PC, KEDIT, THE and the ZOC terminal emulator. Additionally, the Rexx language can be used for scripting and macros in any program that uses Windows Scripting Host ActiveX scripting engines languages (e.g. VBScript and JScript) if one of the Rexx engines is installed.

[1] /blog/polyglot-conway/
[2] /blog/the-developer-guide-to-migrate-across-galaxies/
[3] https://en.wikipedia.org/wiki/Rexx
[4] https://en.wikipedia.org/wiki/Mike_Cowlishaw

The first time I got in contact with Rexx was in 1994, when I installed OS/2 Warp 3.0[5] on my PC, almost 30 years ago. I found it to be a simple, easy to understand language, with some interesting features. It was touted, and rightfully so, as a powerful alternative to standard DOS batch files.

These days, the easiest way to run Rexx programs in Linux or other operating systems is through the Regina interpreter[6], available in Fedora 38 with a simple `sudo dnf install regina-rexx`.

The Rexx language itself is quite simple, and it looks at first glance like a hybrid between BASIC and Python, with some outstanding features:

- Rexx is a fully dynamic language, without declarations of any kind.
- All arithmetic number types are floating-point (like Minimal BASIC[7] and JavaScript)
- Rexx does not have arrays or hashes as other languages, but has "compound variables" that work exactly the same way. Array indexes are directly specified next to the variable name, separated with a dot.
- Regina allows Rexx scripts to use a shebang[8].
- The `say` command always includes a line feed, that's why we use `call charout ,variable` to write to stdout without one.
- Subroutines are simply specified using their name and a colon at the end, like `Evolve:` on like 58, and must have a `return` statement at the end (otherwise the execution continues towards the next subroutine below!) They can have arguments, specified with the `arg` statement at the beginning.
- Although not used on this example, Rexx has an `interpret` instruction that works exactly like JavaScript's `eval()` function.

There's of course a Visual Studio Code extension for Rexx[9], which was very helpful while writing this code. Vim has syntax highlighting for Rexx scripts out of the box.

The Rexx version of my Conway project is available on GitLab[10], and here are the first lines transcribed to satisfy your curiosity:

```
#!/usr/bin/regina

/* Initialization */
size = 29
separator = "|"
generation = 0
do i = 0 to size
```

[5] https://en.wikipedia.org/wiki/OS/2

[6] https://regina-rexx.sourceforge.io/

[7] /blog/conway-in-minimal-basic/

[8] https://en.wikipedia.org/wiki/Shebang_(Unix)

[9] https://marketplace.visualstudio.com/items?itemName=broadcomMFD.lsp-for-rexx

[10] https://gitlab.com/akosma/Conway/-/tree/master/Rexx

```
    do j = 0 to size
        world.i.j = 0
    end
end

signal on halt

call Blinker 0, 1
call Beacon 10, 10
call Glider 4, 5
call Block 1, 10
call Block 18, 3
call Tub 6, 1

/* Print grid in an endless loop */
do forever
    'clear'
    say ""
    do a = 0 to size
        if a = 0 then call FirstLine
        call charout ,format(a, 3)
        call charout ,separator
        do b = 0 to size
            if world.b.a = 1 then call charout ," x |"
            else call charout ,"   |"
        end
        say ""
    end
    generation = generation + 1
    say ""
    say "Generation " generation
    call Evolve
    sleep(0.5)
end

halt:
    'clear'
    return

/* First line with coordinates */
FirstLine:
    do b = 0 to size
        if b = 0 then call charout ,"    "
        call charout ,format(b, 3)
        call charout ,separator
    end
    say ""
    return

/* More: https://gitlab.com/akosma/Conway/-/tree/master/Rexx */
```

COBOL

Do I need to introduce it? Says Wikipedia[11]:

> COBOL (/ˈkoʊbɒl, -bɔːl/; an acronym for "common business-oriented language") is a compiled English-like computer programming language designed for business use. It is an imperative, procedural and, since 2002, object-oriented language. COBOL is primarily used in business, finance, and administrative systems for companies and governments. COBOL is still widely used in applications deployed on mainframe computers, such as large-scale batch and transaction processing jobs. However, due to its declining popularity and the retirement of experienced COBOL programmers, programs are being migrated to new platforms, rewritten in modern languages or replaced with software packages. Most programming in COBOL is now purely to maintain existing applications; however, many large financial institutions were still developing new systems in COBOL as late as 2006.

This is one of the oldest programming languages still in use:

> COBOL was designed in 1959 by CODASYL and was partly based on the programming language FLOW-MATIC designed by Grace Hopper. It was created as part of a US Department of Defense effort to create a portable programming language for data processing. It was originally seen as a stopgap, but the Department of Defense promptly forced computer manufacturers to provide it, resulting in its widespread adoption. It was standardized in 1968 and has since been revised four times. Expansions include support for structured and object-oriented programming. The current standard is ISO/IEC 1989:2014.

This is not my first COBOL code; I first got exposure to the language in April 2020, during the pandemic, when news broke that the governor of New Jersey was desperately looking for COBOL programmers[12]. I got curious and then I enrolled in the Master the Mainframe program by IBM[13], where I learnt a bit about COBOL and JCL[14].

The Micro Focus COBOL[15] and COBOL debugger[16] Visual Studio Extensions were very useful during the writing of this code. Again, Vim has COBOL syntax highlighting support off-the-box.

---

[11] https://en.wikipedia.org/wiki/COBOL

[12] https://nymag.com/intelligencer/2020/04/what-is-cobol-what-does-it-have-to-do-with-the-coronavirus.html

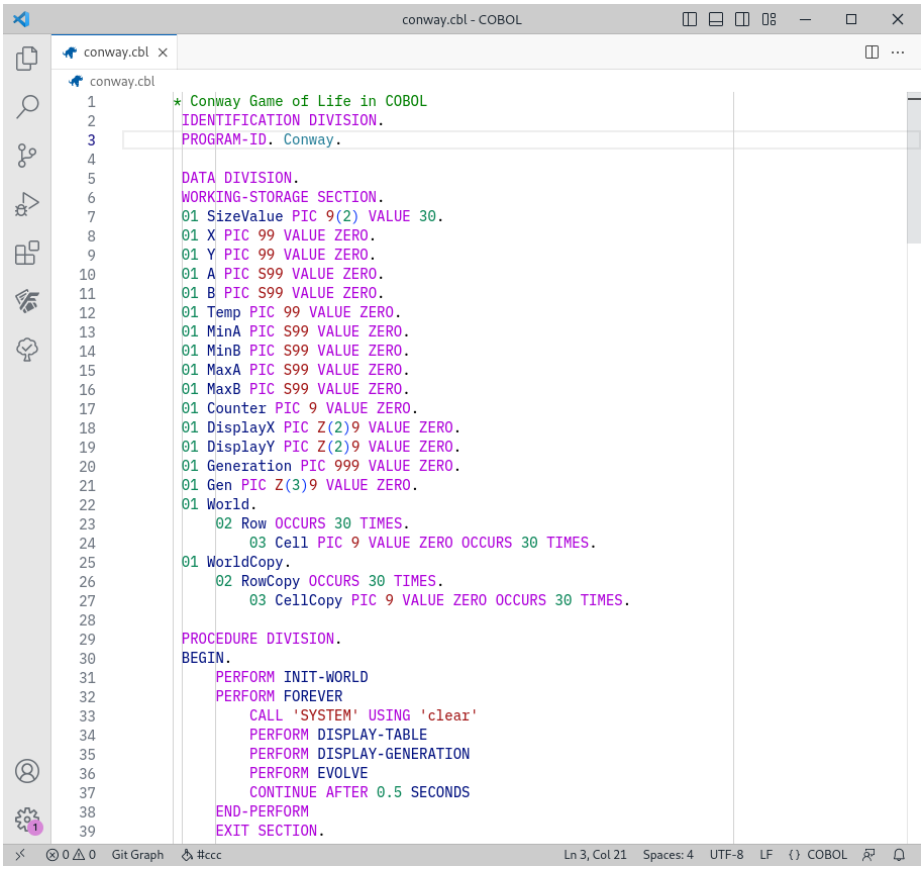
[13] https://en.wikipedia.org/wiki/Master_the_Mainframe_Contest

[14] https://en.wikipedia.org/wiki/Job_Control_Language

[15] https://marketplace.visualstudio.com/items?itemName=Micro-Focus-AMC.mfcobol

[16] https://marketplace.visualstudio.com/items?itemName=OlegKunitsyn.gnucobol-debug

The Micro Focus extension fulfills a very important task: it provides visual vertical guides to properly format COBOL code; **these tabulation limits were mandatory in old COBOL code**, as it replicated the layout of code lines on punched cards. This extension makes it trivial and convenient to quickly navigate those tab stops using the tabulation key on your keyboard:



What else can I say about COBOL?

- Variable types are confusing at first, as they are specified by providing examples of what you want to store inside, leaving the language to decide the best allocation format for them.
  - For example, the statement `X PIC 99 VALUE ZERO` asks to create a variable called `X` that stores positive integers of up to two digits (00 to 99) initialized to zero. The `S` prefix is used to specify that a "sign" is required (hence it can store negative numbers) and the `Z` prefix prevents leading zeros from being printed on the console.
- You don't assign values using the `variable = value` syntax but the `MOVE value TO variable` one.
- You don't `variable += value` but `ADD value TO variable`.
- You don't `variable -= value` but `SUBTRACT value FROM variable`.

- The two-dimensional array used to store the "world" of cells is literally defined with the `OCCURS … TIMES` statement. The compiler very conveniently checks whether there are out-of-bound indexes in your code.
- This program has no I/O, which means that there's no `ENVIRONMENT DIVISION` in this program. The `DATA DIVISION` has only a `WORKING-STORAGE` section, with all the variables used in the program (all global, of course.)
- The code of the program is all contained within the `PROCEDURE DIVISION`, as it should be.
- The `PERFORM` keyword is used to call subroutines by name, and also to loop over variables (`PERFORM VARYING`) or to create infinite loops (`PERFORM FOREVER`).
- The `DISPLAY variable WITH NO ADVANCING` statement just outputs the variable without a line feed at the end.
- The call `CALL 'SYSTEM' USING 'clear'` is obviously non-standard cheating, and should only work on Linux or macOS.

To compile this code I used GnuCOBOL[17], easily installed on Fedora 38 using the classic `sudo dnf install gnucobol` thingy.

The most complicated part of writing this version of Conway was that the documentation is, at best, scattered and hard to find:

- I have a copy of a very good book about COBOL: "Beginning COBOL for Programmers" by Michael Coughlan[18] (2014), but it features code examples written using the Micro Focus dialect of COBOL, which isn't entirely compatible with GnuCOBOL. The book in general is, however, excellent, and I can gladly recommend it.
  - By the way, the source code of the book is available on GitHub[19].
- Although the GnuCOBOL project has very good and decent documentation[20] in PDF and HTML format, it works more like a big manual rather than a quick getting started guide, which is what I would have needed.
- There aren't many questions on Stack Overflow or blog posts to refer to (for obvious reasons!)

I would love to see if this code can run on a mainframe, punching the required cards. A few months ago, on the Vidéothèque section of my magazine De Programmatica Ipsum, I featured a video[21] describing the operation of the wildly successful IBM 1401 computer[22], which was of course used to execute COBOL programs back in the 1960s.

Here's then the first lines of the COBOL version of Conway, available

[17] https://gnucobol.sourceforge.io/
[18] https://link.springer.com/book/10.1007/978-1-4302-6254-1
[19] https://github.com/Apress/beg-cobol-for-programmers
[20] https://gnucobol.sourceforge.io/guides.html
[21] https://deprogrammaticaipsum.com/ken-ross-paul-laughton/
[22] https://en.wikipedia.org/wiki/IBM_1401

on GitLab[23]:

```
      * Conway Game of Life in COBOL
       IDENTIFICATION DIVISION.
       PROGRAM-ID. Conway.

       DATA DIVISION.
       WORKING-STORAGE SECTION.
       01 SizeValue PIC 9(2) VALUE 30.
       01 X PIC 99 VALUE ZERO.
       01 Y PIC 99 VALUE ZERO.
       01 A PIC S99 VALUE ZERO.
       01 B PIC S99 VALUE ZERO.
       01 Temp PIC 99 VALUE ZERO.
       01 MinA PIC S99 VALUE ZERO.
       01 MinB PIC S99 VALUE ZERO.
       01 MaxA PIC S99 VALUE ZERO.
       01 MaxB PIC S99 VALUE ZERO.
       01 Counter PIC 9 VALUE ZERO.
       01 DisplayX PIC Z(2)9 VALUE ZERO.
       01 DisplayY PIC Z(2)9 VALUE ZERO.
       01 Generation PIC 999 VALUE ZERO.
       01 Gen PIC Z(3)9 VALUE ZERO.
       01 World.
           02 Row OCCURS 30 TIMES.
               03 Cell PIC 9 VALUE ZERO OCCURS 30 TIMES.
       01 WorldCopy.
           02 RowCopy OCCURS 30 TIMES.
               03 CellCopy PIC 9 VALUE ZERO OCCURS 30 TIMES.

       PROCEDURE DIVISION.
       BEGIN.
           PERFORM INIT-WORLD
           PERFORM FOREVER
               CALL 'SYSTEM' USING 'clear'
               PERFORM DISPLAY-TABLE
               PERFORM DISPLAY-GENERATION
               PERFORM EVOLVE
               CONTINUE AFTER 0.5 SECONDS
           END-PERFORM
           EXIT SECTION.

      * More: https://gitlab.com/akosma/Conway/-/tree/master/COBOL
```

[23] https://gitlab.com/akosma/Conway/-/tree/master/COBOL

Fortran 95

Finally, a version in Fortran[24], the first time I've ever written any Fortran code:

> Fortran (/'fɔːrtræn/; formerly FORTRAN) is a general-purpose, compiled imperative programming language that is especially suited to numeric computation and scientific computing.

Who created it and what for?

> Fortran was originally developed by IBM in the 1950s for scientific and engineering applications, and subsequently came to dominate scientific computing. It has been in use for over seven decades in computationally intensive areas such as numerical weather prediction, finite element analysis, computational fluid dynamics, geophysics, computational physics, crystallography and computational chemistry. It is a popular language for high-performance computing and is used for programs that benchmark and rank the world's fastest supercomputers.

I used Fortran 95[25] for my code, compiled with GNU Fortran[26] on Fedora 38.

I found it to be a beautiful language to work with; it's quite obvious how the designers of BASIC[27] took inspiration from it. It sadly lost a bit of its edge in scientific computing during the past few decades, in favor of Python and its ecosystem. It's hard to compete against behemoths such as NumPy[28], Project Jupyter[29], and SciPy[30].

But Fortran is still there, still going strong. There's plenty of very good documentation about the language, lots of blog posts, and quite a thriving community of passionate users.

The Modern Fortran[31] Visual Studio Code extension was extremely useful while writing this code, just like Vim and its built-in Fortran support.

Here's a fragment of the full application:

```
program conway
    use world
    implicit none
    integer, dimension(30, 30) :: array
```

[24] https://en.wikipedia.org/wiki/Fortran
[25] https://fortran-lang.org/
[26] https://gcc.gnu.org/fortran/
[27] /blog/conway-in-minimal-basic/
[28] https://numpy.org/
[29] https://jupyter.org/
[30] https://scipy.org/
[31] https://marketplace.visualstudio.com/items?itemName=fortran-lang.linter-gfortran

```
    integer :: generation

    generation = 0
    array = 0
    call blinker(array, 0, 1)
    call beacon(array, 10, 10)
    call glider(array, 4, 5)
    call block(array, 1, 10)
    call block(array, 18, 3)
    call tub(array, 6, 1)

    do
        generation = generation + 1
        call execute_command_line("clear")
        call print(array)
        write(*, '(A)', advance="no") 'Generation '
        write(*, '(I3)') generation
        print *, ""
        array = evolve(array)
        call sleep(1)
    end do
end program conway

! More: https://gitlab.com/akosma/Conway/-/tree/master/Fortran
```

1996

Adrian Kosmaczewski

2023-09-01

My life rebooted in July 1996; the day I decided to drop out from college, to get my driving license, to start going out at least twice a week, and to take a sabbatical from everything. Yes, I pretty much took all of those decisions at the same time.

My life in the summer of 1996 was a mess. I was alone, failing all my exams at the university, and every time I got home, I had to endure the mood swings of an alcoholic mother going through her second divorce.

I didn't have anyone in my life to share all of these things with. I was a rather solitary person feeling like a misfit in most circles. I didn't feel at home with other nerds, I didn't feel at home with non-nerds, and I didn't feel at home… at home.

Tough cookie. Not me, the situation.

So I did the only sensible thing that I could come up with: getting rid of stuff that didn't work.

University was definitely a boring place. I was there stuck studying a subject that I wasn't particularly interested about anymore. So I dropped it. Went to the admissions office (or should I say repudiation office?), signed a paper, walked away.

(Funny story: in spite of not being a student anymore they didn't shut my Internet access down until early 2001, to the point that I published there my first ever personal home page in August of 1996. That website, as humble as it was, would become a key element of my future. But at that time, it was just the coolest thing ever to have: a "web page.")

I loved electronic music, so I started going out. Besides, I wanted to find a girlfriend to hang out with, and I figured out that going out was the best way to reach that goal.

But the most important part was that I missed Argentina. I missed my friends down there, and most importantly, I started asking myself questions about a father I did not know, like, at all.

I went step by step. Going out was far more convenient to do in a car, because Geneva's bus system is quite terrible during the day;

imagine at two in the morning.

Hence the first step was to learn how to drive a car. I had tried learning to ride a motorbike a few years earlier, but two things stood in the way of me reaching that goal: passing the driving exam (which I failed twice) and then not killing myself (which also almost happened.) So I said, get rid of that motorbike, and focus on cars.

Thankfully that decision worked out pretty well. By July 1996 I was borrowing my mother's car every Thursday and Saturday evening, so as to fulfill the second step in my transformation: to go out.

And go out I did. That did not lead directly to the third goal (finding a girlfriend) but hey, can't say I didn't try.

In the meantime I was working at Swissair[1]–no, not Swiss[2], but its older incarnation, the airline that went bankrupt in 2001. By 1996 it was still up and running, filled with hubris and arrogance, but I couldn't care less. I was working 3 to 4 hours a day loading and unloading aircraft in Geneva Airport[3], accumulating hours to be able to buy a discounted plane ticket to Buenos Aires.

I could buy a discounted ticket to any European destination with just 200 hours of work; an intercontinental one (any destination in the world serviced by Swissair, actually) for 400 hours.

And discounted it was: for a round-trip Geneva–Buenos Aires I would only pay 180 Swiss Francs, which was 10% of the standard fare. That's around 210 Swiss Francs or 200 US Dollars in 2023. Of course there was always a catch: that ticket had absolutely no priority, so I would only fly if there was a free seat on the plane after everyone had boarded. The good news was that Swissair was so badly managed, and its tickets prices were so high, that there were always free seats available; in my 3 years as employee of that company, having flown 5 times across the Atlantic and much more around Europe, not a single time did I have the problem of not being able to fly.

(Heck, I even got upgraded once from Buenos Aires to Zurich in Business Class, can you imagine?)

So that's how I landed in Buenos Aires on Wednesday, December 4th, 1996, at around 9 AM local time, to stay in the country for almost two months. That trip changed my life in so many ways that it's worthy of a few more blog posts.

To make a long story short, I fell in love with the wrong girl, and I somehow met my father. Back in Switzerland, I moved out of my mother's flat in March 1997, renting my own place. One of the best decisions I've ever taken.

[1] https://en.wikipedia.org/wiki/Swissair

[2] https://en.wikipedia.org/wiki/Swiss_International_Air_Lines

[3] https://en.wikipedia.org/wiki/Geneva_Airport

In the short 8 months between July 1996 and March 1997 I rebooted my life. In the meantime I learnt HTML, JavaScript, Java, and shortly after that (in October 1997) I stopped handling luggage altogether, and started earning a living writing code running on Netscape or Internet Explorer.

The important takeaway of this article is the following: if you're feeling stuck in your life, if you're unable to move and find a better situation for yourself, you should consider dropping everything. I know, I know; it's easier said than done. And in my case I didn't have many responsibilities back then, either, which made everything easier. But still, I did it, and people should consider the option when the time comes.

PS: I'm about to hit 50 next Monday, hence the introspective post.

Five Years of De Programmatica Ipsum

Adrian Kosmaczewski

2023-09-08

Last Monday I published the 60th edition of De Programmatica Ipsum[1], also known as "DPI", the "unusual magazine about programmers, code, and society, written by humans since 2018." De Programmatica Ipsum has been published continuously every first Monday of each month since October 2018.

In the foreword to the first (and so far, only) compilation of articles[2] in PDF format, Graham[3] accurately described DPI as follows:

> Adrian and I started De Programmatica Ipsum because it didn't exist. We knew about places to get news about the latest programming tools and frameworks. We knew about some amazing bloggers who had really thoughtful content on their favourite—and maybe least favourite—topics. But what's the thing in the middle? Where do we go to get interesting analysis and deep thoughts on varied subjects, to hear from authors we aren't already following, to go beyond "this week in Javascript" and get into the real meaty topics?

This is precisely the spirit that animates this publication. We have tried, thanks to the help of our patrons, to create a different kind of magazine.

The Evolution

Through the years, DPI evolved in various ways:

- During the first year, we featured guest writers whom we paid 150 USD per article. Yes, you read right, we paid our guest writers for their time. Not a lot, and 100% out of our proletarian pockets, but we did it.
- Then we launched the Library section[4], where every month we talk about some book about (or around) software. This section now has almost 50 entries, so if you're looking for book suggestions, we can only invite you to check it out.

[1] https://deprogrammaticaipsum.com/

[2] https://akos.ma/books/De_Programmatica_Ipsum_Vol_1/De_Programmatica_Ipsum_Vol_1.pdf

[3] https://www.sicpers.info/

[4] https://deprogrammaticaipsum.com/category/library/

- Last year we started the Vidéothèque section[5], at this moment with 13 articles, showcasing important video content available on the web, always in the context of the subject of the month.

And DPI will continue evolving in the future.

The Tone

Careful readers will undoubtedly identify the tone of the magazine:

- Left-leaning and absolutely anti-fascist.
- Staunch supporter of worker unions.
- Engaged in a race to highlight the work of non-white, non-male authors in an industry that glorifies male caucasians between 25 and 35 years old, with for example a long list of work created by women[6].
- Fighter against bullshit, ignorance, hubris, arrogance, open spaces, and toxic behavior from all parts of the software engineering spectrum.
- Aiming to highlight the bigger picture, and to provide a social context to the work performed by software developers all over the world. Because, yes, we do believe that we are changing the world with our software; but as members of society, we're not always doing it in the most optimal direction when considering the long term.
- Featuring special editions around August or September entirely dedicated to particular programming languages[7], like those for Smalltalk[8], Java[9], Python[10], Rust[11], and BASIC[12].
- Best viewed with Firefox[13] or LibreWolf[14].

If you are offended by these characteristics (something that, to be honest, goes beyond my understanding) you should probably not read DPI, and actually, fuck off from this website, since you're at it.

The Patrons

Who's paying for DPI? In short, our readers.

We don't track our users, nor clutter our content with advertising. We do not use our newsletter contacts for anything else than to send a new edition every month. The email newsletter itself contains no

[5] https://deprogrammaticaipsum.com/category/videotheque/
[6] https://deprogrammaticaipsum.com/category/women-authors/
[7] https://deprogrammaticaipsum.com/category/programming-languages/
[8] https://deprogrammaticaipsum.com/issue-25-smalltalk/
[9] https://deprogrammaticaipsum.com/issue-24-java/
[10] https://deprogrammaticaipsum.com/issue-35-python/
[11] https://deprogrammaticaipsum.com/issue-47-rust/
[12] https://deprogrammaticaipsum.com/issue-59-basic/
[13] https://www.mozilla.org/en-US/firefox/new/
[14] https://librewolf.net/

advertising of any kind. We do not use affiliate links anywhere on our website.

De Programmatica Ipsum is made with love and with the support of our readers: it is free and ad-free and alive thanks to patronage from readers. We have no staff, no interns, and no assistants. Just two writers, a hosting platform, and a text editor.

If you enjoy DPI, please consider aiding its sustenance with a one-time or recurrent donation. Your support makes all the difference. We do not earn commissions through the links we share to other websites, neither when recommending books, movies, or any other kind of article.

I would like to thank our patrons who generously contribute[15] every month (or have contributed in the past) to our work, and help us run this magazine. Thank you so much! In alphabetical order:

Adam Guest, Adrian Tineo Cabello, Benjamin Sheldon, Christopher Nascone, Franz Lucien Moersdorf, Guillermo Ramos Álvarez, Jean-Paul de Vooght, Patryk Matuszewski, Paul Hudson, Quico Moya, Roger Turner, and Szymon Licau.

Thanks to their constant support, the hosting costs of the website are fully taken care of, the rest of the contributions covering a part of the time spent during the preparation of each issue.

I would also like to thank the guest writers who helped bootstrap this magazine in its first year; again, in alphabetical order:

Adam Jones[16], Adrian Tineo Cabello[17], Agis Tsaraboulidis[18], Anastasiia Vixentael[19], Carola Nitz[20], Daniel Steinberg[21], Fernando Rodríguez[22], Guido de Caso[23], Ioana Porcarasu[24], Julia Cacciapuoti[25], Marie-Cécile Godwin Paccard[26], Roland Leth[27], Sheree Atcheson[28], and Susanna Riccardi[29].

[15] https://deprogrammaticaipsum.com/contribute/
[16] https://deprogrammaticaipsum.com/from-aop-to-ai/
[17] https://deprogrammaticaipsum.com/learnings-from-toxic-abuse/
[18] https://deprogrammaticaipsum.com/the-current-state-of-ethics-in-tech/
[19] https://deprogrammaticaipsum.com/secure-development-is-dead-long-live-secure-development/
[20] https://deprogrammaticaipsum.com/the-creativity-of-computing/
[21] https://deprogrammaticaipsum.com/at-least/
[22] https://deprogrammaticaipsum.com/a-brief-and-painful-history-of-programming/
[23] https://deprogrammaticaipsum.com/on-the-influence-of-programming-language-paradigms/
[24] https://deprogrammaticaipsum.com/on-some-things-quality-related/
[25] https://deprogrammaticaipsum.com/hiring-diversity-beta-version/
[26] https://deprogrammaticaipsum.com/business-as-unusual/
[27] https://deprogrammaticaipsum.com/why-not-both/
[28] https://deprogrammaticaipsum.com/sheree-atcheson-on-diversity-and-inclusion/
[29] https://deprogrammaticaipsum.com/why-i-want-people-to-not-treat-me-differently/

Their outstanding articles have stood the test of time and we invite you to read and share them.

Speaking about which, you can also help us a lot by sharing the posts you like on social media, or by following us on Mastodon[30]. (Do not follow us on our old Twitter account, since our account there has not been used since November 2022, and won't be used in the future.)

But the best of all ways to help us is simple: read our articles. Don't know where to start? Easy: here's a link to a random article[31]. Enjoy!

[30] https://mas.to/@deprogrammaticaipsum
[31] https://deprogrammaticaipsum.com/wordpress/random

Opening Microsoft Access Databases on Linux

Adrian Kosmaczewski

2023-09-15

In the past few months I've been writing about my software archeology experiments, including how to convert old HTML code[1] from 1999 to run in today's browsers, how to run Macromedia Flash movies[2] with Ruffle, or even how to run Java applets[3] now that they aren't supported anymore.

TL;DR: it's possible to open and export Microsoft Access databases in Linux using `mdbtools`.

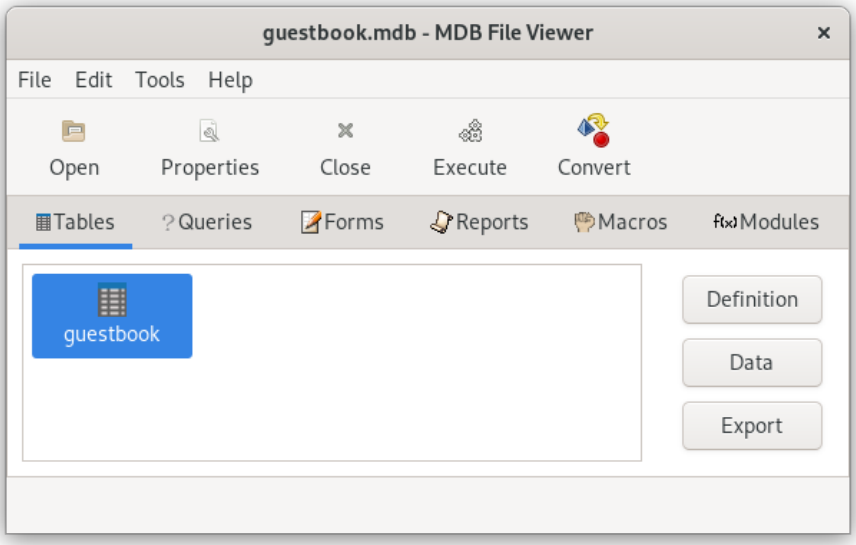
One of the perks of a deep immersion in the Microsoft galaxy is lots of Microsoft Access[4] databases in my archives. To be honest, I actually bought a copy of Microsoft Access 1.0 when it came out in 1992.

As I said in a previous article[5]:

> The backend consisted of Classic ASP[6] pages written in VBScript[7] (that's what I used for work, after all) that stored data in a set of Microsoft Access[8] databases (I know, I know, a really poor choice. Anywaaaaaaaay.)

The MDB Tools[9] project is a set of open source programs meant to read Microsoft Access files from anywhere else than Windows. Even better, there's a GUI built on top called gmdb2[10], which looks like this:

[1] /blog/gamma/
[2] /blog/macromedia-flash/
[3] /blog/java-applets-in-2023/
[4] https://en.wikipedia.org/wiki/Microsoft_Access
[5] /blog/gamma/
[6] https://en.wikipedia.org/wiki/Active_Server_Pages
[7] https://en.wikipedia.org/wiki/VBScript
[8] https://en.wikipedia.org/wiki/Microsoft_Access
[9] https://en.wikipedia.org/wiki/MDB_Tools
[10] https://github.com/mdbtools/gmdb2



The application allows you to export both the structure (as a SQL file) and data (in CSV format) of your Microsoft Access tables, allowing you to create a SQLite[11] file instead, for example (a much better option, no matter how you look at it).

MDB Tools is also used by the KEXI Project[12], the closest thing you'll find to Microsoft Access on the world of Linux.

Even more interesting, the `HACKING` file in the mdbtools project documentation explains the structure[13] of MS Access files in detail.

Installing on Fedora 38

The gmdb2 project can be easily installed in Fedora following these steps:

```
$ sudo dnf install itstool mdbtools-devel gtk3-devel
$ wget https://github.com/mdbtools/gmdb2/releases/download/v0.9.1/gmdb2-0.9.1.t
$ tar xf gmdb2-0.9.1.tar.gz
$ cd gmdb2-0.9.1
$ ./configure
$ make
$ sudo make install
```

After this process, you will have a `/usr/local/bin/gmdb2` executable ready to use. Just `gmdb2 file.mdb` and have fun.

[11] https://sqlite.org/index.html
[12] https://kexi-project.org/
[13] https://github.com/mdbtools/mdbtools/blob/master/HACKING.md

Bonus Track

If you're using Windows and would like to get rid of your Microsoft Access files, you can use this tool[14] to export Access databases to SQLite in one operation.

(Although, to be honest, I don't understand why this tool isn't available for other operating systems as well; after all, it's written in C++ and uses wxWidgets[15] as a GUI toolkit.)

[14] https://github.com/arturasn/mdb2sqlite
[15] https://www.wxwidgets.org/

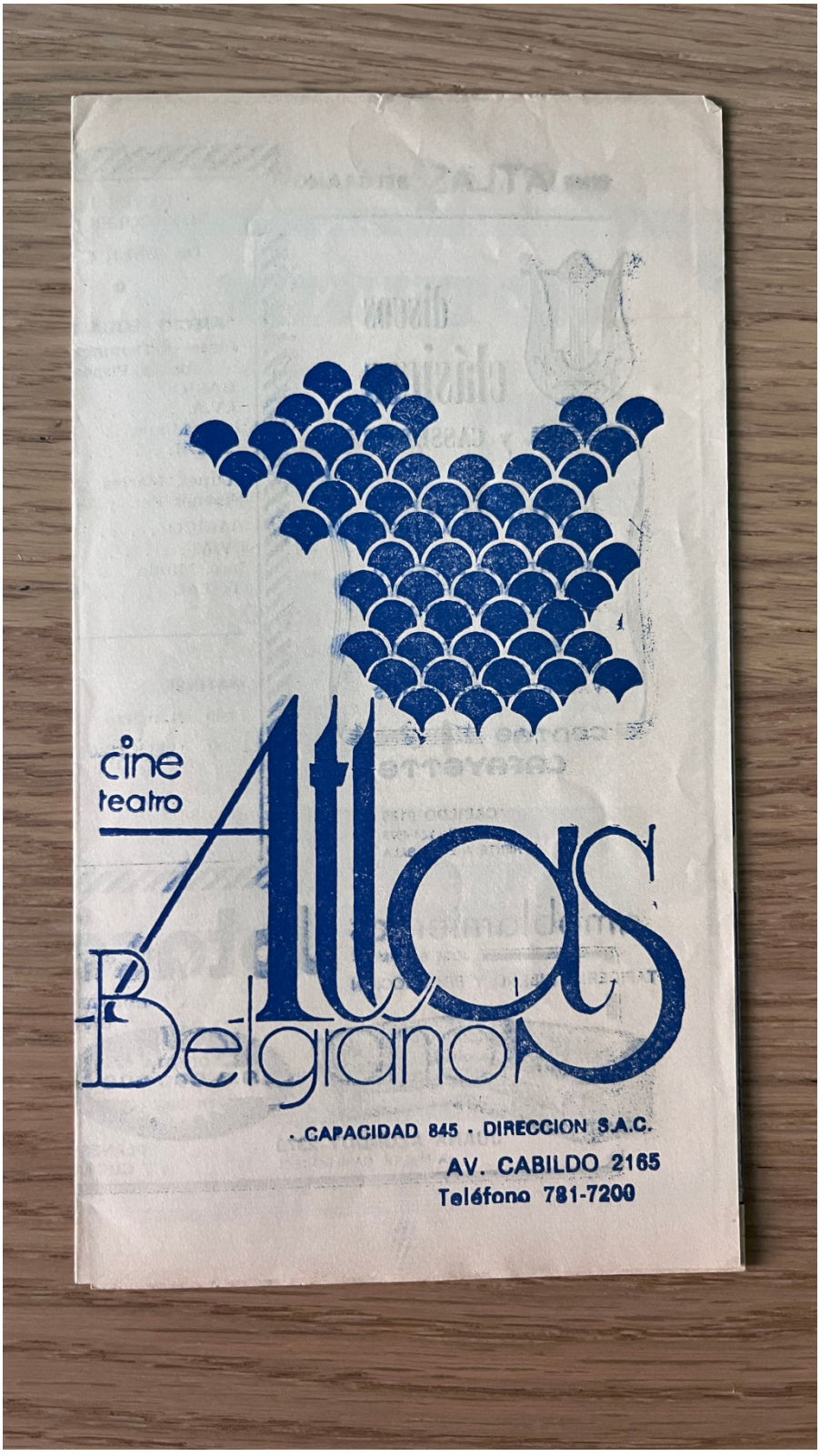
El Imperio Contraataca

Adrian Kosmaczewski

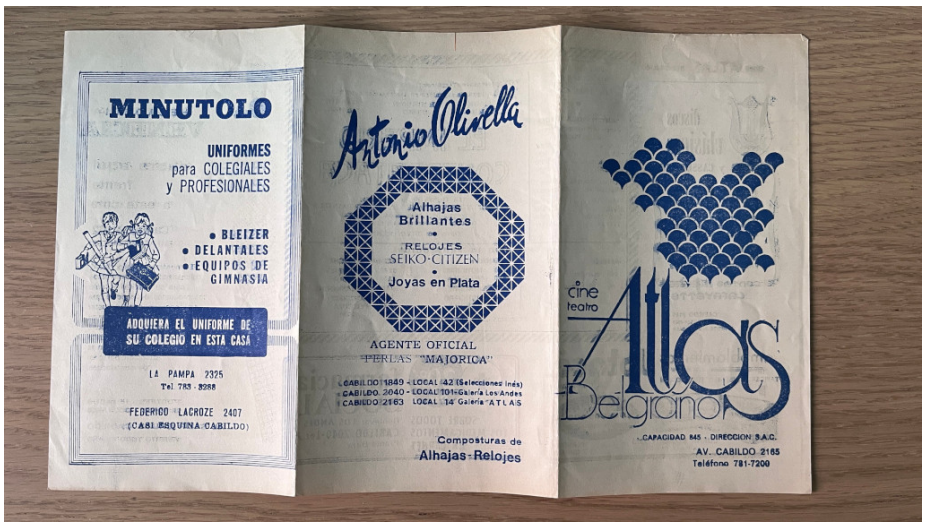
2023-09-22

On the early afternoon of Saturday, January 3rd, 1981, my mother took the 7-year old me to the “Atlas Belgrano” cinema in Buenos Aires (near the corner of Cabildo Avenue and Juramento street) to watch the recently released “The Empire Strikes Back”.

How do I know? I kept the program.


cine
teatro
Atlas
Belgrano
· CAPACIDAD 845 · DIRECCION S.A.C.
AV. CABILDO 2165
Teléfono 781-7200


CINE ATLAS BELGRANO
discos clásicos
y CASSETTES
5000 TITULOS PARA ELEGIR
IMPORTADOS DE: EE.UU. RUSIA - FRANCIA - HOLANDA ALEMANIA - INGLATERRA - Etc.
MUSICA POPULAR
NACIONAL E IMPORTADA
VARIEDAD DE TEMAS
CASA LAFAYETTE
CABILDO 2186
FOX presenta en Tecnicolor:
EL IMPERIO CONTRAATACA
(La Guerra de las Galaxias)
con BILLY DEE WILLIAMS y ANTHONY DANIELS
Dirección: IRVIN KERSHNER
PRECIO LOCALIDADES
HORARIO DE PROYECCION
MATINEE
VERMOUTH
PRIMERA NOCHE
SEGUNDA NOCHE
SABADO TRASNOCHE
Programa Apto para Todo Público
HELADERIA
VENECIA
ahora aquí
frente
a este cine
Cabildo 2192
(SIN SUCURSALES)
En nuestro moderno local, climatizado, le brindaremos para su deleite, lo mejor de los productos de calidad que está acostumbrado a gustar por muchos años en nuestro primer local, recibirá también el trato cordial de siempre, con la atención personal de sus dueños.
POSTRES - HELADOS
SUNDAES - COCKTAILS
AL SALIR CRUCE CABILDO
ABIERTO TODO EL AÑO
amoblamientos
Jotacé
TAPICERIA, DISEÑO Y PRODUCCION
JUANA AZURDUY 2379
$ 3.900.000
farmacia SOCIAL
20% Dto.
SOBRE TODOS LOS MEDICAMENTOS
SIN CARNET
Boulevar LOS ANDES
CABILDO 2040 - Loc. 79
Planta Baja


MINUTOLO
UNIFORMES
para COLEGIALES
y PROFESIONALES
• BLEIZER
• DELANTALES
• EQUIPOS DE GIMNASIA
ADQUIERA EL UNIFORME DE SU COLEGIO EN ESTA CASA
LA PAMPA 2325
Tel 783 - 8088
FEDERICO LACROZE 2407
(CASI ESQUINA CABILDO)
Antonio Olivella
Alhajas
Brillantes
RELOJES
SEIKO-CITIZEN
Joyas en Plata
AGENTE OFICIAL
PERLAS "MAJORICA"
CABILDO 1840 - LOCAL 42 (Selecciones Inés)
CABILDO 2040 - LOCAL 101 Galería Los Andes
CABILDO 2163 LOCAL 14 Galería ATLAS
Composturas de
Alhajas - Relojes
cine teatro
Atlas
Belgrano
CAPACIDAD 845 - DIRECCION S.A.C.
AV. CABILDO 2165
Teléfono 781-7200

Zoom in:

JUEVES 1 AL
MIERCOLES 7

DE ENERO 1981

PRECIO LOCALIDADES

Jueves a Domingo, Feriados y Vísperas

BASICO	$ 8.000
I.V.A.	„ 1.600
Imp. Munic. . .	„ 400
TOTAL	$ 10.000

Lunes, Martes y Miérc. (menos Fer. y Vísperas)

BASICO	$ 6.400
I.V.A.	„ 1.280
Imp. Munic. . .	„ 320
TOTAL	$ 8.000

FOX presenta en Tecnicolor:

La saga de la guerra en las galaxias continúa

MARK HAMILL — HARRISON FORD — CARRIE FISHER

EL IMPERIO CONTRAATACA

(La Guerra de las Galaxias)

con BILLY DE WILLIAMS y ANTHONY DANIELS

Espectacular... grandiosa... La aventura de las aventuras en el espacio.

Dirección: IRVIN KERSHNER

HORARIO DE PROYECCION

Back in those days, in Argentine cinemas, you could get a program of the week, including some local advertising and the classic names of sessions: “matinée,” “vermouth,” first and second night, and on Saturdays, “late night” or “trasnoche.”

That movie cemented my Star Wars fandom member status forever. I remember that session as it was yesterday.

In particular the asteroid field scene[1], by far my preferred sequence across all 11 movies of the whole saga, hands down. Still get the shivers when I listen to the music score. I remember seeing it for the first time that day.

To be honest, it’s one of the greatest moments in science fiction cinema of all time. A triumph. Magnificent. Not even some of the train wrecks that followed (prequels and sequels) could deter me from the saga after that scene.

What. A. Masterpiece.

Anyway.

I don’t remember if we went for an ice cream at the “Heladería Venecia” after the movie, though. January is scorching summer in Buenos Aires, people. Don’t go visit Buenos Aires in January. But ice cream in Buenos Aires is as good as in Rome; of course, because most of the same “gelaterie” in Italy used to have a branch operating in Buenos Aires during the northern hemisphere winter. But I digress.

Also, Argentine VAT was already 20% back in 1981. Christ.

BTW I kept the program in the pages of my copy of the Spanish version

[1] https://www.youtube.com/watch?v=c8deRYotdng

of the novelized version of “The Return of the Jedi” by James Kahn[2].

[2] https://en.wikipedia.org/wiki/Return_of_the_Jedi_(novel)

On Documentation

Adrian Kosmaczewski

2023-09-29

In my career I've seen lots of teams struggling, not only to get their software out of the door, but much more often (even if successful in the previous step) to have a decent level of documentation next to it.

In this article I'll enumerate some misconceptions and myths that I've seen far too many times in the past 26 years I've been in this industry.

I'll also go throught the various levels of documentation that software development teams should keep in their radars at all times. All of these levels are more or less important depending on the context of the software projects; it's not the same to build embedded software for a pacemaker than to build a web application using some fancy enterprise framework. The size of the team also influences the types and depth of documentation that teams should focus on.

Also, this is all based on my admittedly limited experience; I haven't worked at NASA, CERN, nor at a corporation with more than a thousand people in it. There are much bigger software endeavors out there that I haven't been a part of.

Principles

The following are personal beliefs around documentation born out of my own experience; you might agree with, or not. If some or many of the points below sound controversial to you, so be it. I stand by them.

Bad documentation, including bad code comments, is far better than no documentation at all. This is not because of the quality of the documentation per se (after all, "bad" is quite a subjective and time-limited idea that doesn't mean anything at all in the context of software artifacts.) This criteria stems from the following fact: when there is no documentation at all, there is no yardstick to evaluate the quality of the next iteration. The important is not that the documentation is "good" or "bad" because there's no such thing; what there is, always, is room for improvement: if you don't like it, change it. Otherwise, don't complain about it. And if you are in an organization that does not allow you to change it, then move away.

Repeat yourself. The famous "DRY" or "Don't Repeat Yourself" principle is great for your code, but terrible when it comes to documentation. Redundancy in documentation is good, simply because humans learn by repetition, and the only reason you have documentation is because humans must read it to learn about your software. That's why you have it. Hence, yes, repeat yourself, and a lot. Repeat the same concepts again and again, in different places.

It's everyone's task to write docs. This one goes without saying, but for many engineers, "documentation" is a dirty word. Their arrogance makes them consider documentation writing as a lowly kind of work. It is not, and if you have engineers that take this stance, no matter how good they are at writing code, they should either change their attitudes or GTFO. Writing is thinking, and as a software engineer you're paid to think; so, write. I've written about the aversion to writing[1] on De Programmatica Ipsum:

> Around 90% of the teams I have worked with in the past 22 years have never, ever, documented anything. Not a single wiki page, not a README file on top of a repository, not a single PDF file for the end users, not even a single UML diagram. Where they successful? Hardly.

Adopt DocOps. Documentation delivery should be automated, just like all other parts of your system. I'm a big fan of systems like Antora[2] that can be stored on your favorite source code repositories, and then seamlessly integrated into any CI/CD system, including GitHub Actions or GitLab pipelines, so that your documentation is versioned and published automatically after every build of your software. Also, separate content from look and feel (another thing that Antora does superbly well.)

Don't fear drift. It'll happen anyway. Documentation drift (that is, outdated compared to the actual implementation) will happen whether you like it or not. There's no automation that can help you solve this. If you modified the code, you must also modify the corresponding docs. And sometimes you even have to re-read and re-edit the whole documentation package every so often. Yes, it's hard work. Stop whining and get used to it.

Not everything can be automated. This is of course related to the previous point. Very often in my career I've seen teams spending weeks or months to build some contraption that somehow should "automate documentation completely," because humans can't be trusted to fix the drift between code and docs. Such an idea costs more time and money than it would have to just sit down and correct the documentation in the first place. Let's be clear here: **such engineers have the wrong priorities.** Their motivations are understandable, but misguided, and become a net liability for their organization. Most of the time, you just need to sit down and write

[1] https://deprogrammaticaipsum.com/on-the-aversion-to-writing/
[2] https://antora.org/

the documentation. The rest is just brain masturbation that has no room in companies that aren't the size of Google or Meta, which is most of organizations, anyway.

"When you can't create, you can work." This is a quote[3] from American novelist Henry Miller[4][5]. In the case of software engineers, it means that when no more algorithms come to their minds after hours of being in the so-called "zone", it's time to document what they've written during the day. Writing is a muscle; get into the habit of writing down your work every day, and your documentations will naturally flow after a while.

Documentation Supports

Let's talk about the various shapes in which your documentation can come to life.

README Files

This is the first documentation support each and every one of your projects should feature. GitHub invented in 2008 the concept of displaying the README file of a project on its webpage, and since then this practice has spread to GitLab, BitBucket, Gitea[7], and many other similar systems. This, is, without any doubt, one of the biggest contributions of GitHub to the world of software development.

READMEs are the way projects show politeness; they introduce themselves, and they should include at least the following items:

- Purpose: what's this project for?
- IDE-specific instructions: which one to use, how to open the project, which knobs and switches to use to get started.
- Build guidelines, including:
  - Mandatory dependencies.
  - Versions required.
  - Optional tools.
  - Command sequences.
- Quick user's guide: how to use this software?
- How to run tests (on the terminal and on the IDE).
- Any required legal information (licenses, rights, obligations, etc.)

For the sake of brevity, README files can point to wiki pages (more on this later) that expand on these subjects.

[3] https://www.themarginalian.org/2012/02/22/henry-miller-on-writing/
[4] https://en.wikipedia.org/wiki/Henry_Miller
[5] I wrote an article on De Programmatica Ipsum with that title[6] related to freelancing.
[7] https://about.gitea.com/

Code Comments

This is another fundamental element of software documentation. There are lots of schools of thought around this subject, including a whole chapter (32) in Steve McConnell's[8] "Code Complete, Second Edition." I defer the reader to his exposition, clearly worthy of praise.

As a personal trait, I tend to go overboard with comments, usually directing their tone and contents to a future version of myself. I do not think there are useless comments. And if I don't like them for any specific reason, I change them (hopefully for the better) even if I'm not the original author.

There is a particular situation, however, where code comments have become life savers for this author: platform-specific quirks. Every time a piece of software had any tipe of customization required by the different platforms where the code had to run (operating systems, browsers, mobile devices, etc.), comments have always been the best way to document such deviations from the norm.

Serve as examples the specific cases of embedded assembly code in C++ applications, the use of `#ifdef`-type of macros on various programming languages, and of course, specific web browser quirks, particularly during the times of the battle between Internet Explorer versus Netscape and later Firefox.

Wikis

I remember back in 2004 I shocked my (admittedly conservative) peers when I proposed to set up a wiki for our team; to say that I got quite a bit of pushback is the understatement of the year. I persevered, and we finally got a Confluence license… and within hours everybody had poured tons of knowledge in it. It was one of those "why didn't we do this before" kind of moment, and ever since, I could tell whether a company was going to succeed or fail by a simple fact: do they have a wiki or not? It's like an IQ test for organizations.

These days, no need to pay a license of Confluence; you have a wiki bundled into most project management SaaS and self-hosted software; GitHub, GitLab, Gitea[9], Redmine[10], they all have an integrated wiki, usually based on Markdown or some other text-based markup language. So no need to say anything else; just start using it.

Unit Tests

Unit tests are a great documentation mechanism; it's one of the best, actually, particularly for low-level components in an architecture, or

[8] https://deprogrammaticaipsum.com/steve-mcconnell/
[9] https://about.gitea.com/
[10] https://www.redmine.org/

particular APIs, and to understand how they work just by running them.

And if the unit tests have code comments on them, that's even better. And if there are wiki pages talking about the tests, now that's fantastic. Remember, redundancy is your friend.

API Documentation Comments

Most mainstream programming languages offer some kind of API documentation system, used to decorate functions, classes, and whatever constructions you need. These can be later easily extracted using some command-line tool, and put together into a nice website.

The quintessential tool at the origin of such an idea was Doxygen[11], but pretty much every mainstream programming language allows you to do this nowadays (either as a built-in feature, or with a community-provided tool). If your ecosystem offers such a thing, you should definitely add those API comments and then set up a delivery pipeline to publish them for the required teams to see and use.

User-level Documentation

In order to decide the contents and format of this piece of documentation, you should ask yourself: who are the users of your software? All of the conversation, process, formats, and anything else required to create a user-facing documentations derives from that initial question.

The important parts of this type of documentation are, in my opinion, the following:

1. Content: talk to your users in their language; make it easy for them to read your text.
2. Format: provide this documentation in the canonical formats: HTML, PDF, and EPUB are the bare minimum. `man` pages are also a nice touch, depending on your target audience. Good news: Asciidoctor[12] generates all of these formats.

Maintenance Guide

This is a kind of documentation that I used to provide to my customers during the days of my company akosma software[13] (2008-2013). With every application I delivered, I provided a standalone documentation bundle (in PDF and EPUB formats) explaining in detail various aspects of the project (architecture, code organization, IDE settings, etc.) and

[11] https://www.doxygen.nl/
[12] https://asciidoctor.org/
[13] /tags/akosma-software/

how to maintain it in the future. Needless to say, I haven't seen anything similar anywhere else, neither before or after I had my own business.

The idea was for my customers to be completely and totally able to maintain the application after I was gone. Some of them only hired me for the first release, some would come back for subsequent versions; in any case, the maintenance guide was always useful[14].

The inspiration of this maintenance guide came from my father, who is an architect in Buenos Aires specialized in the design and constructions of single-family houses. For every one he builds, he provides his customers with a maintenance guide (in paper format, in the shape of a large binder with typed or hand-written pages) explaining all kinds of details, from the color hues of wall paint, to the recommended brands of plumbing and electricity supplies to be used in case of repairs.

Formats

In what formats should teams write their documentation? First of all, documentation should be stored in textual formats, to enable DocOps workflows; this excludes any use of binary supports such as LibreOffice, Word, or other atrocities.

Most teams start with Markdown[15], and that's usually fine. It's supported everywhere and there are excellent tools to work with it (I'm particularly fond of Typora[16], for example.)

But Markdown is not suitable for complex documentation projects. I strongly recommend using a more evolved markup language, in particular AsciiDoc[17], and even more specific, with the Asciidoctor[18] toolchain.

Asciidoctor provides documentation authors with various invaluable features that Markdown doesn't support off-the-box:

- Built-in page includes.
- Syntax highlighting for code blocks.
- Integration of mathematical equations.
- Support for text-based diagrams.
- Generation of HTML, PDF, EPUB, and `man` pages.

It is also natively supported on GitHub and GitLab, which means that even your README files can be written in AsciiDoc format; just use the `README.adoc` filename and you're done. The Antora[19] documentation

[14] Yes, as counterintuitive as it may sound, my objective was not to lock my users down in working with me. Some of my customers understood the benefits and hired me anyway; those were the best customers I've ever had.

[15] https://en.wikipedia.org/wiki/Markdown

[16] https://typora.io/

[17] https://en.wikipedia.org/wiki/AsciiDoc

[18] https://asciidoctor.org/

[19] https://antora.org/

system I've mentioned previously in this article also stems from the same team that brought us Asciidoctor.

Last but not least, I tend to avoid Textile[20], reStructuredText[21], and other formats[22], and if all else fails, use Pandoc[23] to convert everything to AsciiDoc.

Diagrams

If at all possible, I recommend avoiding proprietary tools and binary formats for your diagrams and to use either (or both) of the following formats instead:

- Standard Vector Graphics, or SVG; for this, Draw.io[24] and its associated Visual Studio Code extension[25] are priceless. Store your files as `filename.drawio.svg` and the Visual Studio Code plugin will load them for you on the editor, automatically.
- Text-based formats, for which I recommend using a self-hosted copy of the Kroki[26] application, which supports pretty much every text-based format you can imagine: BlockDiag[27], BPMN[28], Bytefield[29], C4[30], D2[31], DBML[32], Ditaa[33], Erd, Excalidraw[34], GraphViz[35], Mermaid[36], Nomnoml[37], Pikchr[38], PlantUML[39], Structurizr[40], SvgBob[41], Symbolator[42], TikZ[43], UMLet[44], Vega[45],

[20] https://en.wikipedia.org/wiki/Textile_(markup_language)
[21] https://en.wikipedia.org/wiki/ReStructuredText
[22] https://en.wikipedia.org/wiki/Comparison_of_document_markup_languages
[23] https://pandoc.org/
[24] https://app.diagrams.net/
[25] https://marketplace.visualstudio.com/items?itemName=hediet.vscode-drawio
[26] https://kroki.io/
[27] http://blockdiag.com/en/
[28] https://www.bpmn.org/
[29] https://texdoc.org/serve/bytefield.pdf/0
[30] https://c4model.com/
[31] https://d2lang.com/tour/intro/
[32] https://dbml.dbdiagram.io/home/
[33] https://github.com/stathissideris/ditaa
[34] https://excalidraw.com/
[35] https://graphviz.org/
[36] https://mermaid.js.org/
[37] https://nomnoml.com/
[38] https://pikchr.org/
[39] https://plantuml.com/
[40] https://structurizr.com/
[41] https://ivanceras.github.io/svgbob-editor/
[42] https://kevinpt.github.io/symbolator/
[43] https://tikz.dev/
[44] https://www.umlet.com/
[45] https://vega.github.io/vega/

Vega-Lite[46], WaveDrom[47], WireViz[48], ...

Text-based diagrams can be trivially versioned on any source version control system, and the SVG format guarantees beautiful renderings and impression in every medium, at every resolution.

Fancy Delivery

Of course, most of the documentation tools I've mentioned in this article are web-based. The web offers a fantastic delivery medium for documentation.

These days, however, some new options are appearing, such as the integrated Backstage[49] platform, adopted by Red Hat in their Red Hat Developer Hub[50] system. They offer a powerful mechanism to deliver documentation to your users, and it even considers "technical writers" as an integral part of software development teams.

Conclusion

This was a very long article, so thanks for sticking with me until the end. There's a lot more that could be said, but this article summarizes my thinking around documentation. I hope these ideas will be useful to you too.

[46] https://vega.github.io/vega-lite/
[47] https://wavedrom.com/
[48] https://github.com/wireviz/WireViz
[49] https://backstage.io/
[50] https://developers.redhat.com/products/developer-hub/overview

Learning OpenShift on Linkedin Learning

Adrian Kosmaczewski

2023-10-06

I'm very happy to announce that my new training called "Learning OpenShift"[1] is now available to subscribers at LinkedIn Learning[2]!

[3]

This has been a new experience, supported by the team behind this world-class learning platform called LinkedIn Learning. I hope that the final result will be useful to students interested in learning more about OpenShift[4], the unique and powerful Kubernetes platform created by Red Hat[5].

The course is supported by a GitHub repository[6] containing all the source code shown during the course.

I would like to thank Courtney Allen[7], Simon St. Laurent[8] (with whom

[1] https://www.linkedin.com/learning/learning-openshift/
[2] https://www.linkedin.com/learning/
[3] https://www.linkedin.com/learning/learning-openshift/
[4] https://www.redhat.com/en/technologies/cloud-computing/openshift
[5] https://www.redhat.com/en
[6] https://github.com/LinkedInLearning/learning-openshift-4407066
[7] https://www.linkedin.com/in/courtneyallen2
[8] https://www.linkedin.com/in/simonstl

I had already had the opportunity and pleasure to work when I wrote my two[9] books[10] for O'Reilly), and in particular my coordinator Minyan He[11] for the support during the creation of this training.

For the curious among you, the process from first contact (a private message to Simon after he published a call for teachers[12] on LinkedIn last November) until publication (last Monday) took around 11 months. After proposing a description and title for the course, I had to submit a sample video, to be evaluated by the editorial team, which I did early in December. My course was greenlit shortly before Christmas! I started the preparation of the scripts in February, then the visuals in May, and finally recorded the episode draft videos at the end of June. Minyan provided feedback during each step of the process. The team at LinkedIn remastered the contents adding nicer visuals, animations, and better sound, and they published the final product last Monday evening.

Simultaneously to this release, I'm introducing The OpenShift Guide[13], a website + ebook I created while preparing this course, to be used as support for those interested in OpenShift. The contents of the website (generated with my beloved Antora[14]) can be downloaded as a convenient ebook in PDF, EPUB, or man page formats.

[9] /blog/announcing-my-first-book-mobile-javascript-application-development/

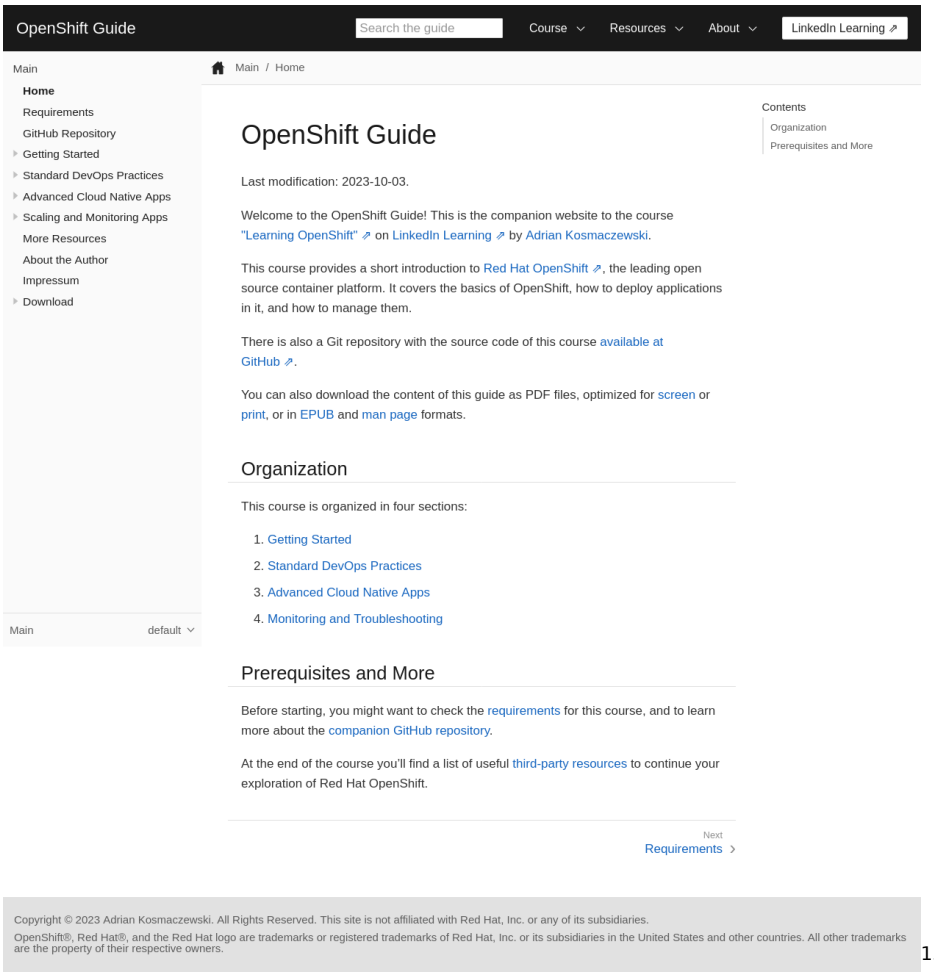
[10] /blog/announcing-my-second-book-sencha-touch-2-up-and-running/

[11] https://www.linkedin.com/in/minyanhe

[12] https://www.linkedin.com/posts/simonstl_tech-instructor-for-linkedin-learning-at-activity-6996561958342664193-cevg/

[13] https://openshift.guide/

[14] https://antora.org/



[15]

[15] https://openshift.guide/

OpenShift Local CRC From Another Machine

Adrian Kosmaczewski

2023-10-13

During the preparation of my "Learning OpenShift"[1] course for LinkedIn Learning, I used OpenShift Local[2] (formerly known as "CodeReady Containers" or CRC) to demonstrate the various features of the platform from a developer's point of view.

The issue with my setup was that, given the resource requirements of CRC, it made my laptop fan run wild, which was a problem since I needed to record my voice as I used the product. To solve this issue, I installed OpenShift Local CRC in a computer in another room connected to my home LAN, and I configured my laptop to access it as if it were running locally using HAProxy[3].

This blog post[4] took me to this gist[5], but the instructions there didn't work on Fedora 38. However, in the comments below the gist there's a reference to this other blog post[6] that actually worked. The final procedure I followed is reproduced here:

On the Server

1. Install HAProxy:

```
$ sudo dnf install -y haproxy
$ sudo systemctl enable haproxy
$ sudo systemctl start haproxy
```

2. Configure HAProxy:

```
$ export CRC_IP=$(crc ip)
$ sudo tee /etc/haproxy/haproxy.cfg &>/dev/null <<EOF
global
    log /dev/log local0
```

[1] https://www.linkedin.com/learning/learning-openshift

[2] https://developers.redhat.com/products/openshift-local/overview

[3] https://www.haproxy.org/

[4] https://cloud.redhat.com/blog/accessing-codeready-containers-on-a-remote-server/

[5] https://gist.github.com/tmckayus/8e843f90c44ac841d0673434c7de0c6a

[6] https://sanjitcibm.medium.com/getting-started-with-openshift-local-f59c07cbfd4c

```
defaults
    balance roundrobin
    log global
    maxconn 100
    mode tcp
    timeout connect 5s
    timeout client 500s
    timeout server 500s

listen app
    bind 0.0.0.0:80
    server crcvm $CRC_IP:80 check

listen apps_ssl
    bind 0.0.0.0:443
    server crcvm $CRC_IP:443 check

listen api
    bind 0.0.0.0:6443
    server crcvm $CRC_IP:6443 check
EOF
```

3. Restart HAProxy:

```
$ sudo systemctl restart haproxy
```

4. Get the IPv4 of the server:

```
$ ip address show
```

On the Client

1. Add the entry below to the `/etc/hosts` file, where `xx.xx.xx.xx` is the IPv4 of the server you got previously:

```
xx.xx.xx.xx api.crc.testing console-openshift-console.apps-
crc.testing  default-route-openshift-image-registry.apps-
crc.testing oauth-openshift.apps-crc.testing
```

2. Open the console on your client machine with the usual URL[7], and enjoy a quiet recording environment, while the fan of your server is buzzing in a separate room.

[7] https://console-openshift-console.apps-crc.testing

Revisiting Ruby on Rails

Adrian Kosmaczewski

2023-10-20

I've blogged about Ruby on Rails[1] quite a few times[2] in the past 18 years. I've delivered lots of Rails apps, I've used it for my own company[3], and I have been a historic fan of Rails against all odds and against all opinions[4]. The levels of productivity I've experienced with it are unparalleled all things considered (and I've tried quite a few web frameworks[5] over the years.)

This is why I'm thrilled to see that not only Ruby on Rails is still alive and kicking, but even better, that it is thriving as it celebrates its 20th birthday[6]!

Imagine that. 20 years. In the software industry it's a very long time. So what's a developer to do to celebrate such a milestone? Well, you guessed it:

```
$ gem install rails
$ rails new app
```

During the process of playing with Rails again I was surprised and delighted to see… how little it has changed! Building an app with Rails is essentially the same today as 20 years ago. Same commands, same reflexes, same high speed. Sign o'the times, the scaffold generator gives you a `Dockerfile` now[7].

I'm watching David Heinemeier Hansson's Rails World keynote[8] as I write this, and three things came to my mind:

1. The original Ruby on Rails demo[9] by DHH (Brazil, 2005) which actually also provided a substantial boost to the sales of TextMate[10]!

[1] https://rubyonrails.org/
[2] /tags/rails/
[3] /blog/running-akosma-software/
[4] /blog/deliver.-now./
[5] /blog/fortune-apps/
[6] https://rubyonrails.org/world
[7] https://www.infoworld.com/article/3706871/ruby-on-rails-extends-docker-support.html
[8] https://www.youtube.com/watch?v=iqXjGiQ_D-A
[9] https://www.youtube.com/watch?v=Gzj723LkRJY
[10] https://en.wikipedia.org/wiki/TextMate

2. Apple's own article about Ruby on Rails[11] in the Developer Connection website, which brought lots of interest on Rails… and on Mac OS X as a serious web development platform (kids, back in those days MacBooks weren't nearly as popular as they are today)
3. DHH kicked off his keynote[12] talking about the evolution of the Federal Reserve interest rates in the past 20 years, and how it impacted the evolution of software frameworks. I think that point was spot-on. I love to see those intersections between society, economy, and our software industry.

Ruby on Rails, boring[13] technology, server-side rendering[14], and the general idea of the monolithic architecture[15], will always have a special place in my heart and in my career as a developer, and I think we're going to be hearing a lot about it in the near future. Here's for 20 more years!

[11] https://web.archive.org/web/20061017073930/https://developer.apple.com/tools/rubyonrails.html

[12] https://www.youtube.com/watch?v=iqXjGiQ_D-A

[13] /tags/boring-tech/

[14] /blog/server-side-rendering-ftw/

[15] /blog/back-to-monoliths/

Notes About “Who Says Elephants Can’t Dance?” by Lou Gerstner

Adrian Kosmaczewski

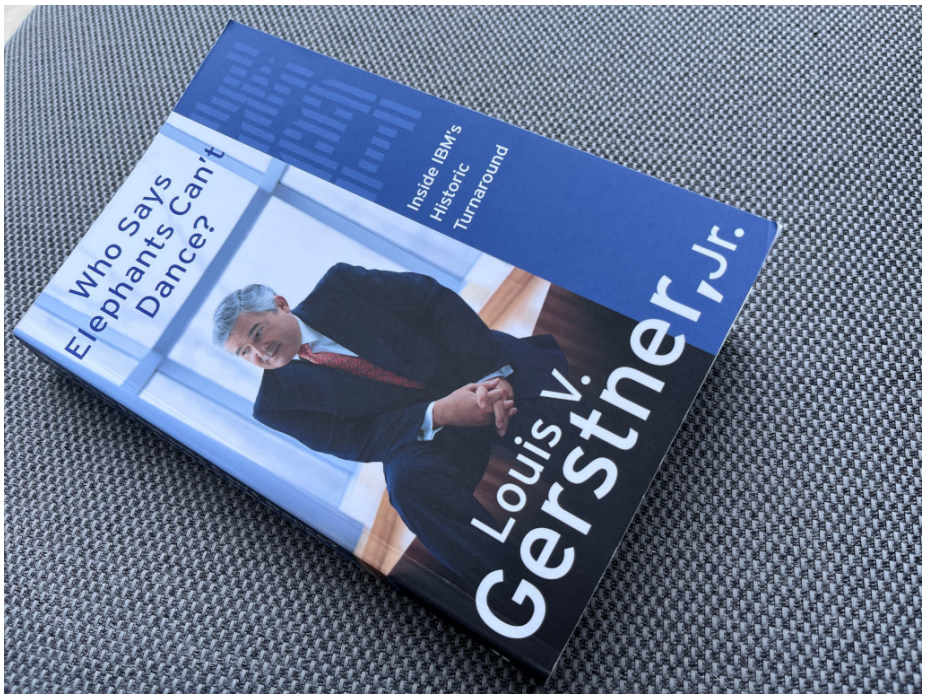
2023-10-27

I finished reading “Who Says Elephants Can’t Dance?” an autobiography of Lou Gerstner, CEO of IBM from 1993 to 2002, and I can say without hesitation that it’s one of the best business books I’ve read in decades.

First, a quote that quite faithfully illustrates the spirit of the book:

> “People who aren’t forced to deal with technology rarely make the effort to understand either its possibilities or its limitations. In the nuclear era, maybe that was all right. But for technologies as pervasive as the ones we’re dealing wich today, I believe we’ll need leaders in government, business, and policy-making roles who commit themselves to the challenge of lifelong learning in order to bring society into sync with the science.”

Here are some notes (quite a few actually!) I took while reading it.



1. Grabbing Hold

- Page 4: meeting “old IBM” and its arrogance as a customer when at AMEX during the 1970s
- Page 5: at Nabisco, learning that “free cash flow” is the single most important measure of business performance
- Page 23: first board meeting
- Page 24: management philosophy principles and five 90-day priorities at the bottom of the page
- Page 32: about his personal assistant Isabelle Cummins becoming a manager of a team of 9 people
- Page 35: meeting with his brother, “brotherly advice”. Also: the mainframe business must not be dismantled
- Page 37: Thomas J. Watson Jr. in the back seat of his car
- Page 40: alcohol policy change in the corporate airplane
- Page 47: switch to customer focus in first meeting with CIOs (mainframe customers) in Chantilly, Virginia (May 1993)
- Page 52: meetings with Andy Grove and Bill Gates
- Page 59: view about the computer industry and need of an integrator
- Page 62: basics of business management
- Page 65: savings
- Page 71: sense of urgency
- Page 72: going back to Watson’s vision but not telling anyone about it
- Page 77: communicate the existence of a crisis to employees
- Page 80: e-mail from a know-it-all

- Page 84: “every institution & individual is a potential IBM customer”
- Page 85: “tech divisions built what could be built without regard for customer needs”
- Page 86: building a customer-oriented organization
- Page 87: manager blocking his emails in EMEA
- Page 89: sales fulfills demand generated by marketing
  - marketing research
  - centralized marketing team
- Page 91: “solutions for a small planet”
- Page 94: compensation & benefits
- Page 96: company should be owned by those working in it
- Page 97: role of stock options in building a sense of ownership
  - 3 reasons in page 98
  - stock ownership mandatory!
- Page 104: “people didn’t mind rebooting their PCs 3 times a day”
- Page 107: “in an industry run by mad scientists, we had to succeed”

2. Strategy

- Page 116: growth of IBM research unit: databases, Fortran, memory chips
- Page 117: culture in an environment without competitive threats
- Page 121: cycles every 10-15 years
  - 1950 (computers) - 1965 (System/360) - 1980 (PC) - 1995 (Internet) - 2010 (mobile & cloud) - 2025 (AI)
- Page 125: role of open standards in a networked world
  - open computing is a threat to companies offering closed architectures
- Page 126: “Post-PC” world
- Page 130: economics of a service business
- Page 132: customers need **integration**
- Page 133: managing services businesses vs product businesses
- Page 137: what is the biggest software company in the world? ~~Microsoft~~ IBM!
- Page 138: OS/2 is a problem
- Page 141-142: the middleware strategy: databases, message queues, transaction management
- Page 142: acquisition of Lotus
- Page 145: acquisition of Tivoli Systems
  - 2001: IBM 2nd biggest software company in the world
- Page 146-152: licensing technology to other companies => IBM research
- Page 156: software strategy, relationship with other ERP vendors, impact on hardware sales
- Page 157: how to make partnerships => contracts, targets
- Page 158: turning competitors into partners
- Page 160: “focus over breadth” => selling the IBM network to

AT&T
- Page 163: “the role of an IT partner cannot be fulfilled by companies that support only one technology or one stack”
- Page 167: “The Cloud” => excellent predictions on page 168
- Page 172: “e-business” => Ogilvy’s ad campaign
- Page 174: “the new economy” => dot-com boom
- Page 177: “The challenge was making that workforce live, compete, and win in the real world” with competition

3. Culture

Chapter 20 is by far the most important chapter in the book.

- Page 182: “Culture is the game”
- Page 183: Description of IBM culture, including “corny” aspects
- Page 184: “Basic beliefs”
  - Excellence
  - Superior customer service
  - Respect for the individual
  - => Dress code changes: abolished in 1995 (page 185)
- Page 185: “We basically acted as if what customers needed had been settled long ago”
- Page 186: Joke: “products aren’t launched at IBM. They escape.”
- Page 187: “Management doesn’t change culture”
- Page 189: “Because IBM was so deeply inbred & ingrown, it had lost its robustness”
- Page 190: “what we could make” was more important than “what the customer needs”
- Page 192: “nonconcurring” option
  - “no one would say yes, but everyone could say no” (page 193)
- Page 195: Four-way matrix at IBM:
  - Geography
  - Product
  - Customer
  - Solutions
  - => Bureaucracy
- Page 199: I had executives, I needed leaders
- Page 200: “high-performance companies are led and managed by principles, not process”
  - Something that would Ray Dalio very happy…
- Page 201: List of Principles:
  1. Marketplace-driven
  2. Commitment to quality
  3. Measure of success is customer satisfaction
  4. Focus on productivity
  5. Strategic vision
  6. Sense of urgency
  7. Teamwork
  8. Sensitive to people & communities

- Page 203: “most people can be roused by the threat of extinction”
- Page 204: “Competitive focus must be visceral, not cerebral”
- Page 206: required behavioral changes
- Page 208: “nothing can stop a cultural transformation quicker than a CEO who permits an executive to disregard the new behavior model”
- Page 210: IBM Leadership Competencies
- Page 211: “win, execute, and team”
  - win: business is competitive
  - execute: speed and discipline
  - team: act as one IBM
- Page 214: “the counterintuitive corporation”

4. Learnings

- Page 220: “company’s focus too narrow in their segment and misses changes in their marketplaces”
- Page 221: IBM proposing to buy Compaq
- Page 222: acquisitions must fit a strategy to succeed
- Page 223: vision ≠ strategy
  - Strategies are based on data
- Page 225: traits of good strategies
  - “believable & executable”
  - “long on detail and short on vision”
- Page 230: Execution is the critical part of strategy
- Page 231: “people do what you inspect, not what you expect”
  - “Accountability must be demanded”
- Page 231: “no credit can be given for predicting rain, only for building arks”
- Page 235: The best leaders create high-performance cultures
- Page 236: Personal leadership => passion
- Page 238: What it takes to run IBM
- Page 239: on integrity: “cultures in which it is easier to ask forgiveness than permission disintegrate over time”
- Page 245: Decentralization or not?
- Page 258: perspective on tech industry: **lack of humanity**
- Page 260: toxic role of investment bankers in industry & economy
- Page 267: “I’ve always believed that less is more when it comes to the press”
- Page 269: “Chasing revenue at the expense of real earnings is one of the most telling signs of a weak management team”
- Page 270: “cash flow, not revenue, sustains corporate success”
  - globalization => overcapacity => commoditization and price deflation
  - success => execution
    * + revenue
    * - costs

Appendix B

- A full prediction of the cloud business that was about to start a few years later!

The Wordpress Market is a Red Ocean

Adrian Kosmaczewski

2023-11-03

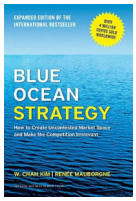
At some point in 2023 I asked online for WordPress experts for the remake of a website, and it felt like opening a fire hose.

In the next few minutes, almost literal seconds, I started receiving not just a few, but literally hundreds of replies. Not only via the online platform that I used at first (LinkedIn) but also via email, instant messaging, and even quite a few phone calls. Yes. Phone calls.

The replies came from all over the world: India, Colombia, Nigeria, Ukraine, Vietnam. They came from small and big agencies, and hundreds of freelance experts requesting for attention. The reaction was so overwhelming that I literally deleted the post one hour later. I could not cope with the quantity of replies.

I did follow up with a few of those offers (arguably, those that arrived in the first few seconds after I published the request) and ended up choosing one of those companies. I explicitly wanted to work with an agency with at least 10 employees.

Some of those agencies I followed up with were extremely unprofessional, while others were doing an admirably good job. The variability in quality of the offers, the handling of the specifications, and the price ranges were outstanding large. In particular regarding the price, the variations were 10x as large from the cheapest to the most expensive. Some companies sent an offer just 24 hours later after our first contact, some did it almost a month after our first contact. Some offers were one-pagers, some were PDF booklets worthy of a multinational.



In 2004 W. Chan Kim and Renée Mauborgne published a (now classic) business book called “Blue Ocean Strategy”[1], together with an article on the Harvard Business review[2]. The central thesis of their

[1] https://www.blueoceanstrategy.com/

[2] https://hbr.org/2004/10/blue-ocean-strategy

work is that some markets are “red oceans” where competition is the toughest you can imagine, with innumerable competitors fighting for scraps.

During the RFP process I understood Kim and Mauborgne’s concepts from the point of view of the consumer: the WordPress market is one of those “red oceans”. It is very hard to tell competitors apart in such markets, if anything because of their sheer number.

I also had the impression that most of these agencies did not realize that they were in such a market; only two or three actively marketed themselves out of this cutthroat environment. I do not know if the other agencies truly understand how much generic they sound and look like from the outside.

The whole idea of a “blue ocean” is to distinguish yourself from the competition in such a way that you truly stand out, as a lone boat would appear in the middle of a blue sea. Your offering stands out, your professionalism shines, your reaction times are swift and short, you go the extra mile at every step of the way, and you distinguish from others in pretty much every possible way imaginable.

In the software industry, the WordPress market is not the only one that requires such tactic for survival; in my career I’ve seen that happen many times: .NET, Android, iOS, “full-stack” web apps, etc. The Kubernetes and Cloud Native markets are becoming red oceans as we speak.

Most of those “red ocean” markets are occupied very early by first movers[3] who enjoy little competition and high margins, and use those profits to grow fast and build a strong position early on. Late-comers can only pretend to scraps, and here is why the creation of a “blue ocean” strategy can make them stand out from the crowd.

It’s not that you should not join an already populated market, but you should do so with a clear market strategy. Most companies don’t even bother, and it shows.

[3] https://en.wikipedia.org/wiki/First-mover_advantage

The Playlist of 2022

Adrian Kosmaczewski

2023-11-10

Here's the playlist of the songs I've enjoyed the most in 2022, aka the playlist of my life. Enjoy!

- 180db_[130][1] by Aphex Twin
- A Rush Of Blood To The Head[2] by Coldplay
- Alone Again[3] by Gilbert O'Sullivan
- Babooshka[4] by Kate Bush
- Believe[5] by Lenny Kravitz
- Big Wheels[6] by Electric Light Orchestra
- Bilder im Kopf[7] by Sido
- Blackbird[8] by Rumer
- Blow It All Away[9] by Sia
- Call it House[10] by Laidback Luke and DJs from Mars
- Carpet Crawlers[11] by Genesis
- Coco Jamboo[12] by Mr. President
- Cold Little Heart[13] by Michael Kiwanuka
- Das Boot[14] by U96
- Dayvan Cowboy[15] by Boards of Canada
- Doo Wop (That Thing)[16] by Lauryn Hill

[1] https://open.spotify.com/track/2VfEcR59Czu8ii3u6kKeP8?si=26767c70c1184756

[2] https://open.spotify.com/track/4JJ5zGKnb1IpERyBrfmb1y?si=026ce1e449754901

[3] https://open.spotify.com/track/4lHQCzdK3VdYQvQZnnRouG?si=35d8654b8b584ea8

[4] https://open.spotify.com/track/53Mz9H5UvWMQUXHZnaZYKQ?si=9b0fb25bc362423f

[5] https://open.spotify.com/track/6x9t21Gl6bRhOrgigur7wK?si=79ab12dba98c4ec4

[6] https://open.spotify.com/track/3T9Yut47mtH2r00TVyT0bi?si=4b98926aafc149d4

[7] https://open.spotify.com/track/3AzVdNe7tCYbjjRzQyVLbN?si=15bd85e5e11e4510

[8] https://open.spotify.com/track/6PLq8twmfpblYKF8y502eY?si=6c71f4b101fe4cae

[9] https://open.spotify.com/track/65Pp8wTUDMWJEVPt6kqd10?si=63a31ea6720f465c

[10] https://open.spotify.com/track/1RE1n34QLdVFmEH178D2bF?si=1f16c074f0834643

[11] https://open.spotify.com/track/439xDVYKceMIrSkrzgYAoX?si=984fbe8fcb3c439e

[12] https://open.spotify.com/track/5fRvePkRGdpn2nKacG7I6d?si=bd69c4fadb684fe7

[13] https://open.spotify.com/track/0qprlw0jfsW4H9cG0FFE0Z?si=b47c54452e7849cf

[14] https://open.spotify.com/track/5A3IdgGphzKS2etiGFB73S?si=0df771b470354ec1

[15] https://open.spotify.com/track/4Y2W4zKa3q72ztbkA0r8Va?si=4ed4d282f1d64ece

[16] https://open.spotify.com/track/0uEp9E98JB5awlA084uaIg?si=fc423cfdbc4940de

- Encore Une Fois[17] by Sash!
- Fooled Around And Fell In Love[18] by Elvin Bishop
- Get Your Head Down[19] by Luke Vibert
- Gualicho[20]
- Handbags & Gladrags[21] by Rod Stewart
- Heroína[22] by Sumo. Luca not dead
- Hit Snooze[23] by High John and Sandro Sáez
- Hob[24] by Jain
- It's My Life[25] by Dr. Alban
- J'ai Demandé à la Lune[26] by Indochine
- Kiss from a Rose[27] by Seal
- Love is Blue[28] by Bon Entendeur and André Popp
- Mas que Nada[29] by Sergio Mendes & Black Eyed Peas
- Mistadobalina[30] by Del The Funky Homosapien
- More Than Words[31] by Extreme
- Oblivion[32] by Astor Piazzolla
- Passing By[33] by Zero 7 and Sophie Barker
- Prince Igor: Polovtsian Dances[34] by Alexander Borodin
- Private Idaho[35] by The B-52's
- Puerto Pollensa[36] by Sandra Mihanovich
- Riverman[37] by Benjamin Clementine
- Santé[38] by Stromae

---

[17] https://open.spotify.com/track/3UPBOL5UtCZmJRwioMkfLD?si=bd3555b7bee4401c

[18] https://open.spotify.com/track/2hE5Lm5XOHR4t3xlhlFauP?si=47772a5483b74718

[19] https://open.spotify.com/track/4SYVoeYf6tMll3e0UHKlnU?si=6e780538ea73423e

[20] https://open.spotify.com/track/2OQpt8JOqTFct1nbdyUifv?si=d92d01c551d2474b

[21] https://open.spotify.com/track/1ByOQkAzOlq7XXlBJ7EZce?si=dd36e862c27f4df3

[22] https://open.spotify.com/track/7ERzS9KlggtCp9oslPzEc0?si=79969b68eea74836

[23] https://open.spotify.com/track/314RevBqs4yM87xsFFhAA1?si=c343b5d1ea9f4e83

[24] https://open.spotify.com/track/0OLUGU7cl7NrhOJLfHAkI1?si=47acf552a59a4762

[25] https://open.spotify.com/track/6HdM7gzXVgcpepv276raog?si=2dad13ec9e904a4c

[26] https://open.spotify.com/track/2QSAj76Ba6aMFX9RlXdUdO?si=14c619ab9f004273

[27] https://open.spotify.com/track/3YKptz29AsOlm7WAVnztBh?si=8ef86c25d9a1444a

[28] https://open.spotify.com/track/0FYbUa5RbUKwSH5oPghUsu?si=fa0fcab9c63e4e0e

[29] https://open.spotify.com/track/6U7GUjtamt2P0LcFod1dBT?si=71fc66b8de984e92

[30] https://open.spotify.com/track/0KQd3QY1Y8zC2rfO4ZBQRC?si=68835e19b5f94a03

[31] https://open.spotify.com/track/2SbsvihBJmgNKZ4qkXFWrp?si=2e7313377132412c

[32] https://open.spotify.com/track/0CShxQaSFFKXY2PxrJIhrB?si=62e02ecbf7bf4bf3

[33] https://open.spotify.com/track/7fGAChEASw3sxA1TFRhPfx?si=71f01b8a82e14d9b

[34] https://open.spotify.com/track/6nfgWKfyDfFglNd8zn7gxq?si=a52d96088fc04d4b

[35] https://open.spotify.com/track/0Y87qqNyXLEwv7VNBQ6Thq?si=177ebc6c934f4076

[36] https://open.spotify.com/track/5BaoWORszQDCU5ee4frtoP?si=46bc136c28054928

[37] https://open.spotify.com/track/5qhbi7a0ogsvyCmxhekvxe?si=0f3d87ff3bff4cdb

[38] https://open.spotify.com/track/3vXnuFnC5RhPGwsFi0ORcI?si=553d6b4b4d00401b

- Signals[39] by Cinnamon Chasers
- So What'Cha Want[40] by Beastie Boys
- Sound and Vision[41] by David Bowie
- Streetlife Serenader[42] by Billy Joel
- Summerhead[43] by the Cocteau Twins
- Symphony in Blue[44] by Kate Bush
- The Bomb (These Sounds Fall Into My Mind)[45] by The Bucketheads
- The Heat Is On[46] by Glenn Frey
- The Ladder[47] by Röyksopp (and in general the whole album Profound Mysteries[48] is a triumph!)
- The Real Slim Shady[49] by Eminem
- Tic, Tic, Tac[50] by Carrapicho
- Un Soffio Caldo[51] by Zucchero
- Under the Bridge[52] by Red Hot Chili Peppers
- Visa från Utanmyra[53] by Jan Johansson
- Walk Like an Egyptian[54] by The Bangles
- Wetlands[55] by Nora en Pure
- When I'm Small[56] by Phantogram
- Woman in Chains[57] by Tears for Fears
- Y Lo Que Quiero Es Que Pises Sin El Suelo[58] by Catupecu Machu

[39] https://open.spotify.com/track/68Zv3xIbYSkDFMK80lUi1t?si=45cf5e0215d0436a

[40] https://open.spotify.com/track/0gmGBCJ5XhOmoNR37MmxEE?si=29dbbcb27a97464f

[41] https://open.spotify.com/track/1vP2JEXRsGrFbwOZ0foOQ5?si=8047ed676b3c489c

[42] https://open.spotify.com/track/5tFUPBJqdtWjxCdBtqDwRf?si=4651c485f8474ee4

[43] https://open.spotify.com/track/2mGAozbOumOSTvEEd76Cp5?si=9c4e046d200e4da2

[44] https://open.spotify.com/track/1ADI3yWnCH2cgZE9y6Ss0j?si=1cb45af64e94464a

[45] https://open.spotify.com/track/2Biy0tRkZTGxGH8axDwnqP?si=d754d7f47e5c4447

[46] https://open.spotify.com/track/0PXw9NKvolWTo7U9JkNzmc?si=b22f5972499a42f0

[47] https://open.spotify.com/track/3rnHE4lsm2YvqNq7NizPHH?si=537ba06d883b476e

[48] https://open.spotify.com/album/3yNkCirqEtZOU2gKQhUjVA?si=EH4egLB-QkuB2_ermppqpA

[49] https://open.spotify.com/track/3yfqSUWxFvZELEM4PmlwIR?si=7bb0c9fa76f14ee7

[50] https://open.spotify.com/track/33FrM0i2g7uosARZ4aCFk1?si=7ab22c4db3a848f3

[51] https://open.spotify.com/track/7ziEXEMvMIO00BL9Uha0fh?si=8eb8ebe5af6f4341

[52] https://open.spotify.com/track/3d9DChrdc6BOeFsbrZ3Is0?si=16467195216248c6

[53] https://open.spotify.com/track/1CRxANVlNyN2IYjarx96gU?si=6b1aa0651f294537

[54] https://open.spotify.com/track/1Jwc3ODLQxtbnS8M9TflSP?si=4bc923587957494b

[55] https://open.spotify.com/track/3DTdC9y1dLT1BmfV8mgKKi?si=958c2ad45cc944d5

[56] https://open.spotify.com/track/0PERS23jNsjcjBywXLvpyc?si=30089c3eb4b74d9c

[57] https://open.spotify.com/track/5jmwL2MniQj9ldptIm1h1B?si=5059a684699a454b

[58] https://open.spotify.com/track/3z5F0EaY9VCq9GiQPzrbDX?si=7650166bbb624374

The Core Idea of this Blog is Migrations

Adrian Kosmaczewski

2023-11-17

This blog, made of writing pieces I published online in various formats since 1996 to today, is the reflection of the various migrations I've experienced through my life.

I've been through two kinds of migrations: physical and technological. Look at them; they're grouped under the "migrations" tag[1].

There's the story of my moving from Buenos Aires to Geneva[2], and the one about dropping macOS for Linux[3]. The other about seeing .NET developers switching to Rails[4]. There's even a tongue-in-cheek guide[5] for those who want to move to another technical galaxy, by the way. And I keep learning a new programming language every year[6] since 1992.

This is the core idea of being a polyglot software developer. I always felt like I couldn't stop moving in this industry, that I had to keep up with whatever came next. And that's why so many posts in this blog reflect new things, new discoveries, product reviews[7], experiments[8], explanations[9], and even papers[10], some of which I wrote during the time of my Master's degree.

All things change continuously. And I'm naturally curious, so I'll keep on writing about migrations on this blog.

---

[1] /tags/migrations/
[2] /blog/thirty-years/
[3] /blog/migrating-from-macos-to-linux/
[4] /blog/migration/
[5] /blog/the-developer-guide-to-migrate-across-galaxies/
[6] /blog/yup-still-learning-a-new-programming-language-every-year/
[7] /tags/product-reviews/
[8] /tags/experiments/
[9] /tags/explanations/
[10] /tags/papers/

Expecting People to Work for Free

Adrian Kosmaczewski

2023-11-24

Business literature is filled with myths and legends about famous entrepreneurs building empires from scratch. What those same books don't say is that those empires are often built on top of free labor.

I know of lots of very well-known entrepreneurs in the local Swiss market (don't sweat it out, won't name names) whose "empires" were built that way. I know it because I was often asked to provide free work to those same mavericks.

Let's enumerate some anecdotes:

- Before the dotcom boom of the 2000s, knowing HTML was a huge competitive advantage. No wonder lots of wannabe founders contacted me to help them build a new "portal" or whatever the fancy name du jour was. Of course, as soon as I asked for compensation, they literally got offended and walked away.
- The same happened during the 2008-2013 iPhone craze, where knowing how to put an app together triggered the same kind of emotions on people. I remember one of them in particular, the maker of a relatively well-known web app who wanted a mobile version, and wanted me to work for free, and he'd pay 5% of each sale of the final app. Great deal indeed. Thanks, but no thanks.
- Another notable case was a person who apparently had invented a replacement for XML and wanted me to help him market the concept, without compensation, by the way. Not even equity, nothing. As soon as I mentioned that XML did the same as his thing, and surprise surprise, it already existed, he got angry with me because "I couldn't see the potential." Oh well, I guess I couldn't.
- I had once an interview with the creator of a very important system (yes, you've heard of it and probably also about the person) who was starting something in the mobile space and got my reference. He explained his idea to me for a while, and at the end I said, "yes, it's interesting, my hourly rate is..." and then he interrupted me, saying, "oh no, I hoped you'd worked for free for me." Befuddled, I asked, "why would I do that?" "Well," he replied, "don't you know who I am?"
- And let's not forget about the myriad of "friends" who got angry

with me because I didn't want to work for free to make their ideas happen. Yes, during the times of the web and mobile hype trains, I had plenty of friends, you know.

Thankfully I wasn't active on the blockchain or AI galaxies. I'm pretty sure I would have had quite a few more anecdotes of their kind to share here.

In retrospect, few of the systems I enumerated above ever saw the light of day, and none exist today. What remains is the bitter taste in your mouth of having to deal with a very toxic group of people, half-assed "entrepreneurs" who try to squeeze as much juice from workers to try to build some meaningless useless thing during a gold rush.

I'm not saying that all entrepreneurs are like this, and thankfully I've also met a lot that have had seriously good ideas, were brilliant at execution, and they are still thriving after all of these years. But I met too many of those in the larger section of the Gauss curve, useless opportunists, unbearable to deal with.

5 Years of Full-Time Linux

Adrian Kosmaczewski

2023-12-01

Five years ago I bought a TUXEDO Computers[1] laptop and finished my migration to Linux completely, without ever looking back.

The whole migration process was a planned, much longer process, one that started almost two years before, when I lost patience[2] with Apple and its shenanigans.

I documented most of the aspects of the migration in this blog[3].

I started working as a professional software engineer in 1997 using a desktop computer running Windows 95, writing ASP pages in VBScript and running websites on Windows NT and SQL Server 6.5. Then at some point during the 2000s I migrated to the Apple galaxy, mostly to write iOS applications[4]. And these days I spend my time with Linux kernels, containers, and OpenShift[5] clusters.

Putting all of those experiences in the linear scale of time, I found these to be the proportions I spent in each galaxy[6] so far:

- 39% Windows
- 42% Mac
- 19% (and growing) Linux

In the past 5 years I have used Linux exclusively, both at work[7] and for my own personal computing needs and projects. I've dived into Linux as much as I could: trying out various package managers[8], switching from Ubuntu[9] to Fedora[10], backing up my system with Restic[11], playing with GNOME extensions[12], doing some video production[13] on

[1] https://www.tuxedocomputers.com/en
[2] /blog/courage/
[3] /blog/migrating-from-macos-to-linux/
[4] /blog/running-akosma-software/
[5] /blog/learning-openshift-on-linkedin-learning/
[6] /blog/the-developer-guide-to-migrate-across-galaxies/
[7] https://vshn.ch/
[8] /blog/linux-package-manager-strategy/
[9] /blog/ubuntu/
[10] /blog/fedora-38/
[11] /blog/restic/
[12] /blog/gnome-extensions/
[13] /blog/recording-getting-started-guides-on-linux/

it even if it sucks[14], and even opening Microsoft Access databases[15] with it.

Was it worth it? Yes, totally. Linux is a solid, stable, simple operating system where most things just work. It's not annoying nor hypocrite as their commercial counterparts, and it doesn't stand in my way to get things done. Does it crash? Yes, as much as macOS and Windows. Does it annoy me? It's far from perfect, but it doesn't even come closer to what I had to endure with macOS and Windows.

To put it in simple terms, it underpromises and overdelivers.

[14] /blog/video-editing-in-linux/
[15] /blog/opening-microsoft-access-databases-on-linux/

30 Years of Product Naming Trends

Adrian Kosmaczewski

2023-12-08

Here is an extension of a popular tweet[1] I once wrote, extending it both backward and forward in time, with some actualizations.

1994: eProduct
1997: Product!
2000: Product.com
2003: 37products
2006: Productr
2009: Product.app
2012: Product.io
2015: Productify
2018: ProductCoin
2021: Product.ai
2024: ProductGPT

[1] https://twitter.com/akosma/status/706579358132146176

Microcelebrities

Adrian Kosmaczewski

2023-12-15

My wife is a yarn geek. She knits beautiful garments, but she also dyes and sells her own yarn on her online shop[1]. We often go to trade shows, with a literal truckload of hand-dyed yarns, to sell her products. And in those events I discovered that there are "microcelebrities" in the yarn world, just like there are in the software world.

A few weeks ago I wrote about[2] my encounter with one of those microcelebrities:

> I had once an interview with the creator of a very important system (yes, you've heard of it and probably also about the person) who was starting something in the mobile space and got my reference. He explained his idea to me for a while, and at the end I said, "yes, it's interesting, my hourly rate is…" and then he interrupted me, saying, "oh no, I hoped you'd worked for free for me." Befuddled, I asked, "why would I do that?" "Well," he replied, "don't you know who I am?"

My wife has had her share of "don't you know who I am?" in the yarn world as well. In our world of social media and followers and podcasts, I guess all fields of human activity must have their fair share of microcelebrities nowadays.

I have been a microcelebrity in my own right, too. I remember going to conferences, back in the days of akosma software[3], and having people come to greet me, ask questions about some programming language, and even to be featured in interviews[4] on TV and print and whatnot.

Having a few thousand followers in some network like TikTok or Instagram allows you to start charging for product promotions and more. There's a lot of money to be made… and a lot of arrogance to be displayed.

[1] https://isalloni.com/

[2] /blog/expecting-people-to-work-for-free/

[3] /blog/running-akosma-software/

[4] /tags/interviews/

The Last Day of a Tech Conference

Adrian Kosmaczewski

2023-12-22

Have you ever stayed until the end of a conference, and walked down the halls of the conference center during the last day of an event?

Have you ever felt that eerie sensation of the last afternoon, when most attendees and exhibitors have already left? When security guards don't even check badges anymore and are just wandering in front of their assigned rooms? Have you seen those empty food stands and lunch tables? Have you seen those people feverishly working on their laptops on a couch at the far end of a huge empty hallway?

I've seen this in two particular places: at the Red Hat Summit in Boston last May, or at the four WWDC I have attended in San Francisco in 2008, 2009, 2010 and 2012. The bigger the conference center, the more striking the effect.

The few people remaining are maybe attending some workshops scheduled for the last day, scattered across the vast halls of the venue. This sensation is particularly daunting in those huge US conference centers, capable of holding events for thousands of people at once.

Most of the stands on the exhibition hallways are emptied on the evening of the last official day of the event, but some crews are still working on the remaining ones, usually the biggest of them all, requiring cranes and a few more people than usual to wind the logos down to the floor.

The few remaining people are scattered, particularly after the last lunch (if any) is over. Some are working on their laptops, some are waiting for a cab, and some are meeting someone else in extremis, right before taking off.

And then a cab or a limo takes you away, too.

Remembering SETI@Home

Adrian Kosmaczewski

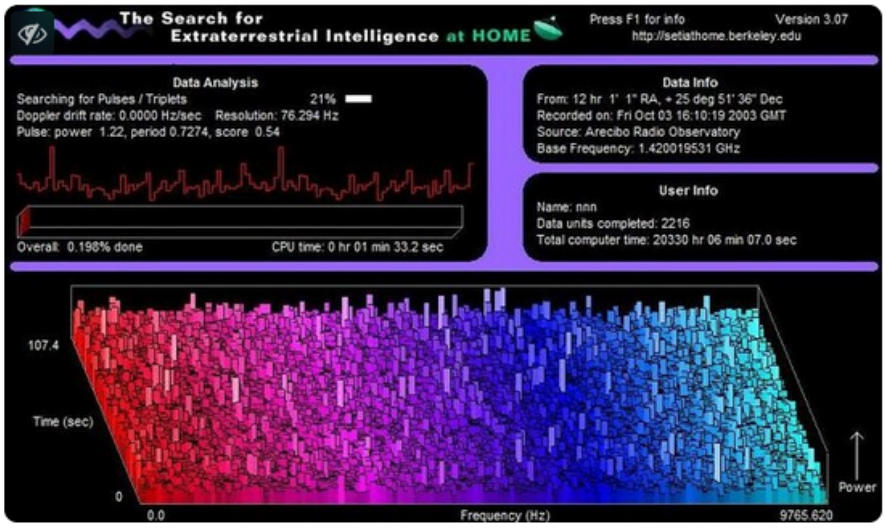
2023-12-29

I recently saw a toot[1] on Mastodon about the end of the SETI@Home program… and it brought back memories of the late 1990s.

Andrew Leahey
@andrew@esq.social

Wow! Apparently as of 2020 SETI@home is "in hibernation" -- end of an era if this is/was the end. I remember running their screensaver on my 386 in 1999 and dreaming about having enough processing power to actually make a dent.

RIP if this is it. Screenshot not mine: GPL, commons.wikimedia.org/w/index.…



Mar 18, 2023, 01:58 · · Ivory for Mac · 61 · 106

[2]

[1] https://esq.social/@andrew/110041571456351699
[2] https://esq.social/@andrew/110041571456351699

SETI@home[3] was quite an idea back in the 1990s: donate some CPU time, process some radiotelescope data on your PC, and you might help scientists find whether E.T. is out there phoning us. Of course I ran the software quite a bit, maybe even too much for the taste of my boss at the time, who found the whole idea inane and useless.

These days, instead of looking for extraterrestrial life, people use their GPUs to generate cryptocurrencies. Sign o' the times.

[3] https://en.wikipedia.org/wiki/SETI@home