Being A Developer After 40

Adrian Kosmaczewski

2016-04-25

This is the talk I gave at App Builders Switzerland on April 25th, 2016.

The slides are available on SpeakerDeck and at the bottom of this article. The video of the session is available in YouTube. This article has been printed in the June 2016 edition of Hacker Bits.

Thanks to some awesome translators, here are some localized versions of this article:

-
- Magyar
- Čeština
- Português
- Persian—
- Vietnamese—Tiếng Việt
- Românesc

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:

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, just north of Buenos Aires, Argentina. The moment was Unix Epoch 876142800. 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. 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. Steve Jobs had taken the role of CEO and convinced Microsoft to inject 150 million dollars 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" had just started. Gianni Versace had just been murdered in front of his house. Mother Teresa, Roy Lichtenstein and Jeanne Calment (the world's oldest person ever) had just passed away. People were playing Final Fantasy 7 on their PlayStation like crazy. BBC 2 started broadcasting the Teletubbies. James Cameron was about to release Titanic. The Verve had just released their hit "Bitter Sweet Symphony" and then had to pay most royalties to the Rolling Stones.

Excite in 1997, courtesy of the Internet Archive

Smartphones looked like the Nokia 9000 Communicator; 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. Not as many apps but the battery life was much longer.

IBM Deep Blue had defeated for the first time 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," 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, Chloë Grace Moretz and Kylie Jenner were born.

Many film storylines take place in 1997, to name a few: Escape from New York, Predator 2, The Curious Case of Benjamin Button, Harry Potter and the Half-Blood Prince, The Godfather III and according to Terminator 2: Judgement Day, 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 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, to HotMeTaL Pro to EditPlus, managing cross-browser compatibility between Netscape Navigator and Internet Explorer 4, and writing stored procedures in SQL Server 6.5 powering a commercial website published in Japanese, Russian, English and Spanish—without any consistent UTF-8 support across the software stack.

The product of these efforts ran in a Pentium II 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, SQL Server 6.5 and IIS 2.0, serving around ten thousand visitors per day.

My first professional programming language was this mutant called VBScript, 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 to understand how to do anything in JavaScript.

Unit tests were unheard of. The Agile Manifesto 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+ component in Visual J++ to parse JPEG files uploaded to the server. As soon as JPEG 2000-encoded files started popping up, our component broke miserably.

We did not use source control, not even CVS, RCS or, God forbid, SourceSafe. Subversion did not exist yet. Our Joel Test 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. I was featured in Swiss and Bolivian TV shows. I watched live keynotes by Bill Gates and by Steve Jobs in Seattle and San Francisco. I spoke at and co-organised conferences in four continents. I wrote and published two books. 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 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 & RUP.

In 2000 it was SOAP & XML.

In 2003 it was Model Driven Architecture and Software Factories.

In 2006 it was Semantic Web and OLPC.

In 2009 it was Augmented Reality.

In 2012 it was 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, you know.

> The mobile phone evolution [@ValaAfshar](https://twitter.com/ValaAfshar?ref_src=twsrc%5Etfw) pic.twitter.com/ShP206GiYL
>
> — JM Alvarez-Pallete (@jmalvpal) November 26, 2015

The stories surrounding the genesis of our industry are fascinating, and will show you two things: first, that everything is a remix. 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 by Michael A. Hiltzik
- Revolution in the Valley by Andy Hertzfeld
- The Cathedral and the Bazaar by Eric S. Raymond
- The Success of Open Source by Steven Weber
- The Old New Thing by Raymond Chen
- The Mythical Man Month by Frederick P. Brooks Jr.

You will also learn to value those things that stood the test of time: Lisp, TeX, Unix, bash, C, Cocoa, Emacs, Vim, Python, ARM, GNU make, man pages. 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
>
> — Hisham (@hisham_hm) December 13, 2015

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? Be my guest. Design? Of course. UX? You must. Poetry? You should.

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 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 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 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 by Tom DeMarco and Tim Lister
- The Psychology of Software Programming by Gerald M. Weinberg
- Facts and Fallacies of Software Engineering by Robert L. Glass
- The Design of Everyday Things by Don Norman
- Agile!: The Good, the Hype and the Ugly by Bertrand Meyer
- Rework by Jason Fried and David Heinemeier Hansson
- Geekonomics 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 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 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

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, 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 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 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

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 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."

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 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" 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

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 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 and Thomas Piketty. 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

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.

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

> Desirable developer skills:
>
> 1 Ability to ignore new tools and technologies 2 Taste for simplicity
> 3 Good code deletion skills 4 Humility
>
> — David Winterbottom (@codeinthehole) December 3, 2014

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,) Dropbox Pyston, the Clang Static Analyser, ARC, Google Souper, Emscripten, LLVMSharp, Microsoft LLILC, Rubymotion, cheerp, watchOS apps, the Android NDK, Metal, 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 with it, and Mono already uses it. Facebook has tried to integrate LLVM with HHVM, and WebKit recently switched from LLVM to the new B3 JIT JavaScript compiler.

LLVM is cross-platform, 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](https://twitter.com/owensd?ref_src=twsrc%5Etfw)
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

10. Follow Your Gut

I had the gut feeling .NET was going to be big when I watched its introduction in June 2000. I had the gut feeling the iPhone was going to be big when I watched its introduction in 2007.

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

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 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.

The impossible roundabout

Follow your heart Follw yur heart Fllw yr hart Fw y art Fart

— Daniel Kibblesmith (@kibblesmith) November 17, 2014

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.

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, Umberto Eco, Gato Barbieri and Johan Cruyff passed away, or when SQL Server was made available in Linux, or when Google AlphaGo beat a Go champion, or when the Panama Papers and the Turkish Citizenship Database were leaked the same day, or when Google considered using Swift for Android for the first time, 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 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.

Slides

Video