The Developer Guide to Migrate Across Galaxies

Adrian Kosmaczewski

2017-04-28

This is the presentation I gave at the second App Builders Conference in Lausanne, Switzerland, April 25th, 2017.

- 1. Perl, Python, Ruby, and PHP (aka Babelia)
- 2. VBScript (aka Apocalypsia)
- 3. C++ (aka Paradigmia)
- 4. Objective-C (aka Coolia)
- 5. JavaScript (aka Undefinedia)
- 6. Swift (aka Migrania)
- 7. Local Cluster (aka Here, Now)
- 8. Guardians Of The Galaxy (aka You)
- Slides
- Video

Last year I briefly touched the concept of "technological galaxies" in my best-seller talk of all time, the one I gave in Zurich and which you have most probably recommended or shared on social media.

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 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)

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 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 of an independent software vendor claiming that they are "artisanal" and "organic." This is typically what hipsters do.

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, and you are not friends with John Gruber, 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.

Dongle Galore

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 had not yet happened and you still had your Walkman plugged to your ears listening to New Order.

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 or AbiWord can open; well, almost.

The beauty of the Equation Editor in 2017 opening files from 1992

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.

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, an implementation in Perl made by Gruber himself, with contributions from the late Aaron Swartz and others.

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. Somebody used Python to make reStructuredText and AsciiDoc. A guy from Colorado used Ruby to recreate AsciiDoctor.

And then a philosophy professor, the king of all hipsters, used Haskell to make Pandoc and brought everything under the same roof. Even Word, LibreOffice,

AbiWord and man pages, AsciiDoc and Textile, Markdown and reStructured-Text, PDF and RTF, HTML and XHTML, all of them together in just one tool. How about that.

2. VBScript (aka Apocalypsia)

Apocalypsia

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.

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.

The book that never was

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. 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.

> 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

    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, 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)

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 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."

The beauty of C++ template error messages

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.

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)

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 his days in a clinic in the Alps, to treat his 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 his 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 his 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.

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)

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.

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 his 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.

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)

Migrania

One of the most brilliant aspects of both the JVM and the CLR is that they are, by definition, Abstract Binary Interfaces. 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 Mi-*

crosoft 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 saying that you could actually use COM in Mac OS X… 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#, JRuby, Kotlin, Scala (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 you can even transpile those .NET assemblies you built with C# into native executables for iOS.

The name of the LLVM project means "Low Level Virtual Machine." And Swift is arguably the first language born from the project, one 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. 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

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)

Local Cluster

I have changed galaxies quite often in my career.

As I said last year:

> 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?
>
> what I find interesting is their convergence.
>
> — akosma (@akosma) April 3, 2017

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

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 these days.

Java and C++ have gotten optionals recently, also called "option types," "Maybe" or "Nullable" just as many other languages.

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.

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](https://twitter.com/dotnet?ref_src=twsrc%5Etfw) pic.twitter.com/Js8wgmx7ER
>
> — Scott Hanselman (@shanselman) April 13, 2017

Crazy! Geez.

8. Guardians Of The Galaxy (aka You)

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.

Pseudoscrum.

Open Spaces.

Harassment.

Lack of testing and quality.

Burnout and depression.

Π ( ) 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 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.

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.

Slides

Video