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) 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 and his own blog, Theocacao.

From the former, suffice to mention gems like the "Learn Cocoa" 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" book; I then read the massive "Cocoa Programming" 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" 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 and Graham Lee 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 language known as Swift.

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 and John Sundell.

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.

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 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.