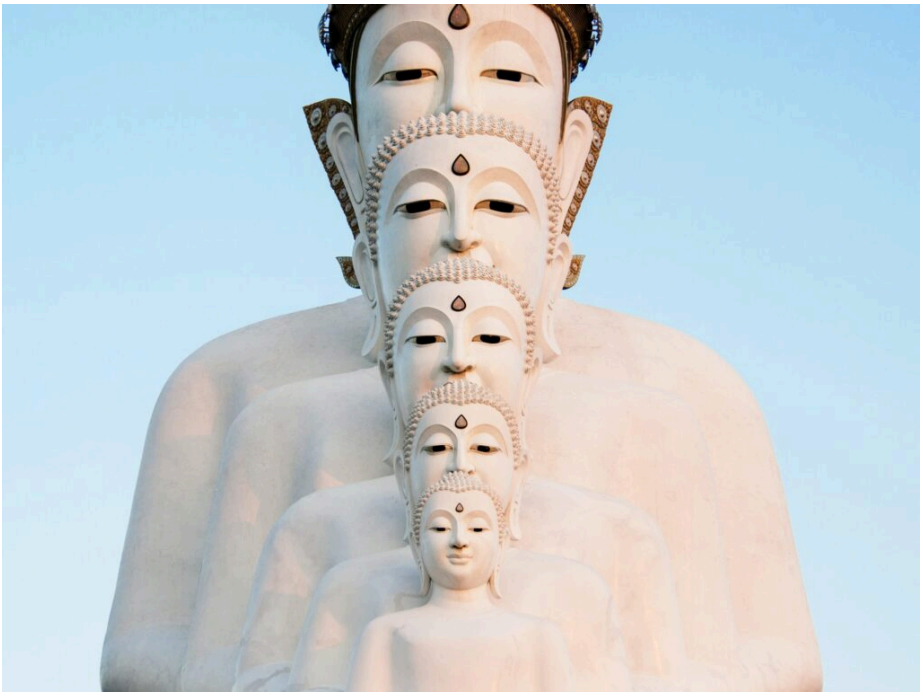
Issue 020: Cycles

Graham Lee

May 4th, 2020



Welcome to the twentieth issue of *De Programmatica Ipsum*, dedicated to the subject of *Cycles*. In this edition:

- Adrian argues that cycles are unavoidable[1] and a strict characteristic of our industry.
- On the other hand, Graham thinks that cycles are instead based on perception[2] and familiarity.
- In the Library section[3], Adrian writes about Kathy Sierra[4] and her work in the *"Head First"* book series.

Enjoy this issue! Please subscribe to our free newsletter[5] to stay updated about new releases, or contribute[6] if you would like to support our work.

Cover photo by Céline Haeberly[7] on Unsplash[8].

[1] https://deprogrammaticaipsum.com/eternally-finally/
[2] https://deprogrammaticaipsum.com/there-aint-nothin-new-under-the-sun/
[3] https://deprogrammaticaipsum.com/category/library/
[4] https://deprogrammaticaipsum.com/kathy-sierra/
[5] https://deprogrammaticaipsum.com/newsletter/
[6] https://deprogrammaticaipsum.com/contribute/
[7] https://unsplash.com/@celinehaeberly?utm_source=unsplash&utm_medium=referral&utm_content=creditCopyText
[8] https://unsplash.com/s/photos/repetition?utm_source=unsplash&utm_medium=referral&utm_content=creditCopyText

Eternally Finally

Adrian Kosmaczewski

May 4th, 2020

It is hardly possible these days to browse any blog or news website commenting the latest trends in the Apple galaxy, without reading references to the mythical, yet entirely foreseeable, ARM-based Mac. The existence of this not-yet-announced piece of otherwise overpriced hardware is entirely predicted by a closer reading of technology history in that same galaxy, an exercise usually (sadly) abhored by many.

In the past four and a half decades, Apple has moved its products across successive CPU architectures in a continuous flow. Looking at it from the vantage point of this dystopian future of ours, each of the products in the cover picture of this article show a product with a different CPU architecture. After the 8-bit MOS Technology 6502[1] that powered the Apple II around 1980, Macs used the 16-bit (later 32-bit) Motorola 68000[2] around 1990, then 32-bit (later 64-bit) IBM PowerPC[3] around 2000, later settling with Intel's 32-bit x86[4] and 64-bit x86-64[5] around 2010.

We are *in* 2020, not even *around* 2020 anymore, and if we believe all of those blogs, we should be able to spend an insane amount of money to buy[6] Macs powered by ARM architecture CPUs[7] – "*finally!*", as some of those headlines online would scream. After all, even the latest iPhone SE models are beating[8] those old crusty Macs in pretty much every Geekbench test out there. And even Microsoft is selling ARM-based laptops[9] these days.

[1] https://en.wikipedia.org/wiki/MOS_Technology_6502
[2] https://en.wikipedia.org/wiki/Motorola_68000
[3] https://en.wikipedia.org/wiki/PowerPC
[4] https://en.wikipedia.org/wiki/X86
[5] https://en.wikipedia.org/wiki/X86-64
[6] https://www.bloomberg.com/news/articles/2020-04-23/apple-aims-to-sell-macs-with-its-own-chips-starting-in-2021
[7] https://en.wikipedia.org/wiki/ARM_architecture
[8] https://daringfireball.net/2020/04/the_2020_iphone_se
[9] https://www.microsoft.com/en-us/p/surface-pro-x/8vdnrp2m6hhc?activetab=overview

For each one of those CPUs enumerated above, there has always been an "official" development environment; from Integer BASIC[10] for the Apple II, to Object Pascal[11] for the Mac, to C++ and MacApp[12], to CodeWarrior[13] and PowerPlant[14], to Objective-C[15] and Cocoa[16], and finally to Swift[17] and the upcoming SwiftUI[18]. These "sacred" environments were always historically considered the "one true way" to build software for those platforms. Steve Jobs' famous "Thoughts on Flash"[19] open letter from 2010 still exemplifies that hard stance. Other means are frowned upon, or simply downright banned.

As an interesting pattern, these five programming languages have all been sufficiently different from one another as possible; thereby reducing the chances of mutual compatibility to an absolute minimum. The differences among them are not only of syntactic nature, which would have been a lesser problem, but in terms of runtime behaviour, memory models, and module patterns; the whole Swift application binary interface affair has recently put these difficulties in perspective once again.

In each step of the way, to ensure a certain smoothness in the migration to a new architecture, Apple has provided some sort of emulation or transitional technology, allowing software from a previous era to run unmodified in the next, or at least to be easily recompiled.

This has always happened, and will happen again.

Examples abound. The Apple IIe Card[20] allowed Apple II software to run on the Macintosh LC[21] in the early 90s. The Mac 68k emulator[22] enabled Mac apps written for the 68000 family to run on PowerPC CPUs from 1992 to 2008. From 1994 to 1998, the Macintosh Application Environment[23] made Mac apps run on A/UX[24], Apple's flavor of Unix before Mac OS X. The Classic Environment[25] allowed PowerPC applications to run on Mac OS X from 2000 to 2008. The Carbon API[26] allowed developers to "recompile and run" older Mac OS 9 apps to run on Mac OS X from 2000 to 2012. Rosetta[27] allowed Mac apps built for the PowerPC architecture to run on Intel CPUs from 2006 to 2011. More or less during the same timeframe, the Universal Binary[28] executable format allowed the same bundle to run natively in both Intel and PowerPC CPUs.

And as I write these lines, during one of the most complex times of our era, I hear the governor of New Jersey desperately looking for COBOL programmers[29], prompting a

[10] https://en.wikipedia.org/wiki/Integer_BASIC
[11] https://en.wikipedia.org/wiki/Object_Pascal#Early_history_at_Apple
[12] https://en.wikipedia.org/wiki/MacApp
[13] https://en.wikipedia.org/wiki/CodeWarrior
[14] https://en.wikipedia.org/wiki/PowerPlant
[15] https://en.wikipedia.org/wiki/Objective-C
[16] https://en.wikipedia.org/wiki/Cocoa_(API)
[17] https://developer.apple.com/swift/
[18] https://developer.apple.com/xcode/swiftui/
[19] https://www.apple.com/hotnews/thoughts-on-flash/
[20] https://en.wikipedia.org/wiki/Apple_IIe_Card
[21] https://en.wikipedia.org/wiki/Macintosh_LC
[22] https://en.wikipedia.org/wiki/Mac_68k_emulator
[23] https://en.wikipedia.org/wiki/Macintosh_Application_Environment
[24] https://en.wikipedia.org/wiki/A/UX
[25] https://en.wikipedia.org/wiki/List_of_macOS_components#Classic
[26] https://en.wikipedia.org/wiki/Carbon_(API)
[27] https://en.wikipedia.org/wiki/Rosetta_(software)
[28] https://en.wikipedia.org/wiki/Universal_binary
[29] https://www.cnbc.com/2020/04/06/new-jersey-seeks-cobol-programmers-to-fix-unemployment-system.html

seven time increase[30] in the number of downloads in the GnuCOBOL project. And since it thankfully supports various COBOL dialects, such as COBOL85, X/Open, COBOL2002, COBOL2014, MicroFocus, IBM, MVS, ACUCOBOL-GT, RM/COBOL, and BS2000, developers can just start up their copy of Visual Studio Code in your laptop of choice, with Intel or ARM CPUs, and ask for access to a test mainframe[31]. The COBOL cycle is starting up again.

Yes, Jean-Baptiste[32], vous aviez raison; **plus ça change, plus c'est la même chose.** Yes, Heraclitus[33], you were right too; *πάντα χωρεῖ καὶ οὐδὲν μένει*. Everything stays the same, because the only constant is change. Yes, Peter[34], *everything old is new again*.

It is no surprise that Raymond Chen[35] chose the a title such as "The Old New Thing" for his blog[36] (and book[37]) about the history of Windows.

As a corollaire, it is actually easy to predict that in the 2030s Apple (and, *de facto*, the rest of the industry) will once again migrate their product lineup to some other architecture, with a new fancy programming language featuring lambdas and monads and objects and most probably created with LLVM[38] and (somehow) understandably incompatible with its predecessor. There will be some kind of transitional technology for both users and developers. A new breed of YouTube stars and bloggers will rave about this (old) new thing and make you feel old and forgotten once again.

This all will happen more or less at the same time COBOL will once again be fashionable, as we approach the year 2038[39]. Because cycles is what the computer industry essentially is made of.

Cover photo by the author.

[30] https://sourceforge.net/projects/open-cobol/files/stats/timeline?dates=2020-03-01+to+2020-04-25
[31] http://ibm.biz/cobollabs
[32] https://en.wikiquote.org/wiki/Alphonse_Karr
[33] https://en.wikiquote.org/wiki/Heraclitus
[34] https://en.wikipedia.org/wiki/Peter_Allen_(musician)
[35] https://devblogs.microsoft.com/oldnewthing/author/oldnewthing
[36] https://devblogs.microsoft.com/oldnewthing/
[37] https://www.pearson.ch/Informatik/Addison-Wesley/EAN/9780321440303/Old-New-Thing-The
[38] http://llvm.org/
[39] https://en.wikipedia.org/wiki/Year_2038_problem

There Ain't Nothin' New Under The Sun

Graham Lee

May 4th, 2020

A tutorial on some scientific software package –I don't remember what– reminded me that it's easy to see familiarity in novel settings. The author of this tutorial saw three environments worth describing in the context of trying to use this software. Linux and Windows were two. The third is UNIX. UNIX, the author explained, is a venerable and robust operating system with a long heritage. The modern context in which you would see UNIX is on a Mac.

Of course, this author is correct. macOS – well, some specific versions that have been submitted for certification – officially is UNIX, the Open Group own the trademark and they say so, because Apple pays them to say so. But many Macintosh aficionados will be up in arms. UNIX is an implementation detail, macOS is so much more than that: it's the windowing system, the user experience, the graphics subsystem, the Swift programming language, the App Store, the integration with phones, tablets, and the cloud: it's a whole philosophy of computing, of which UNIX is one legacy implementation detail.

Still others will be abhorred. Macs are just round plastic children's toys, they may have some warmed-over Mach hybrid as a bootstrap but they aren't really UNIX systems. A true – or should I say TRU? – UNIX can only be found on a computer you can't lift. And yes, that includes the laptops: those Tadpole SPARCbooks[1] aren't light.

We saw the same thing in the 1993 movie *Jurassic Park*. When Lex exclaims "It's a UNIX system! I know this!" she is not focusing on the same part of the computer experience as much of the audience. The user interface looks less like the then-standard Common Desktop Environment than a fancy HollywoodOS[2] installation. It is, in fact, an advanced window manager from Silicon Graphics. Other appearances of common operating systems in movies, including the installation of Linux on Dillinger's glass desk between Tron and Tron: Legacy, tend not to get commented on in-universe.

[1] http://www.computinghistory.org.uk/det/36028/Tadpole-SPARCbook-3GX/
[2] http://wiki.c2.com/?HollywoodOs

What all of these analyses have in common is that we look for the familiar to help us understand the novel. macOS is indeed a UNIX, or a NeXT Workstation, or a Macintosh, or a mass-market triviality. What we need to understand is how the novel parts fit in, to expand our capabilities. It's a UNIX *and*.... It's a Macintosh *and*....

Thing about the *and* is surprisingly hard. Alan Kay talked about the *creation* of ideas in terms of the pink and blue planes. The pink plane is where incremental improvements to existing paradigms, practices, and technologies are found. The blue plane, being orthogonal to the pink plane, is where departures from the status quo live. This is where break-out ideas come that redefine our expectations, open up entirely new fields of endeavour, and let us do entirely new things rather than doing the same things more efficiently.

Kay unfairly puts all of the effort onto the inventor, and all of the failing onto everybody else: it's up to Alan to have the great idea, and you're all bozos for not getting it. As we've seen, letting the novel in along with the familiar is hard, and it'd be easier if we worked together. The inventor needs not only to have the idea, but to demonstrate its possibilities and benefits. As we have seen, Object-Oriented Programming and Smalltalk did not take off just because they were good ideas, but because Adele Goldberg[3] and others promoted, developed, and explained the ideas.

Also, the audience needs not only to listen to the idea, but to be receptive to the idea that it contains something new. Any subculture contains geeks, MOPs, and sociopaths[4], and it's going to be the geeks to turn to here. They're the ones who see the new idea, the development, the radical departure, when everyone else sees a repeat of last year's news. When you look at a new music player and think "No wireless. Less space than a nomad. Lame."[5], there are others who see a new way of experiencing music.

Technology is only cyclic when you choose to experience it as a cycle.

Cover photo by Tadd and Debbie Ottman[6], CC BY 2.0, via Wikimedia Commons[7].

[3] https://deprogrammaticaipsum.com/adele-goldberg/

[4] https://meaningness.com/geeks-mops-sociopaths

[5] https://slashdot.org/story/01/10/23/1816257/Apple-releases-iPod

[6] https://www.flickr.com/photos/30304782@N00/2664190034/in/photolist-KyhSg-6UDoKx-54qESW-54qEWw-5CW2mT-4ToetS-7zMGdi-2gdbZgt-cE6QK1-o2dqN-5BVs4Z-6haJC8-4SGHid-81Lxku-81LwVs-6VFwco-3BqXFZ-81Lx79-3Buujo-5tftcP-KyhSe-hRZZu-m3wS4-47PmvB-4FUTiM

[7] https://commons.wikimedia.org/w/index.php?curid=85470173

Kathy Sierra

Adrian Kosmaczewski

May 4th, 2020

The history of programming language books can be roughly divided in three distinctive eras.

The first one stretches from the beginnings of programming to the mid 1970s. Programming books from those times were an often underestimated byproduct of the marketing budget of big companies such as IBM, and inherited the dry approach of most engineering books in the post-war era. They served their purpose well, although they can be hardly remembered. We have covered one such example in a previous edition[1] of this series; suffice to say that the treatment of these big tomes targeted different times, and a different mindset.

An interesting, enduring, albeit late example of this first era could be the classic "Algorithms + Data Structures = Programs"[2] by Pascal's creator, Niklaus Wirth[3]; a recommeded read, worthy of its own entry in this series.

The second era can be called the "Hello, World!" era. It began with the venerable Kernighan & Ritchie's "The C Programming Language"[4]. During that era, most programming books, or dare I say all of them, included some kind of "Hello, World!" demo in the first chapter. Getting those words through the console is the first contact most developers had with most programming languages. Stroustrup's book about C++[5], Dave Thomas' about Ruby[6],

[1] https://deprogrammaticaipsum.com/jean-sammet/
[2] https://en.wikipedia.org/wiki/Algorithms_%2B_Data_Structures_%3D_Programs
[3] https://en.wikipedia.org/wiki/Niklaus_Wirth
[4] https://en.wikipedia.org/wiki/The_C_Programming_Language
[5] http://www.stroustrup.com/4th.html
[6] https://pragprog.com/book/ruby/programming-ruby

Kochan's on Objective-C[7], and Armstrong's about Erlang[8], all contain a "Hello, World!" exercise in the first few chapters, or some variation thereof; I will leave to the reader the exercise of finding other references.

The structure of these programming books are well know to everyone in the field; a first chapter highlighting key elements of the syntax, the mechanisms and commands required to compile and run the code, and an increasingly complex series of subject chapters towards the end. Text and code, rinse and repeat.

I admit being a culprit in this respect, having written at least one book[9] with such an approach. Let my most sincere apologies be known to those who survived the experience.

Some might argue that the third era was the "Effective" era, with books such as Meyers' Effective C++[10], Bloch's Effective Java[11] or Wagner's Effective C#[12] as flagship examples. But I consider these books, as useful as they are, just an offspring of the second era; a much needed upgrade. Let's say, then, it was the 2.5 era of programming books.

The third era actually started with the publication of "Head First Java"[13], and got its first major bestseller with "Head First Design Patterns"[14] in 2005. In this era, books are no longer dry sequences of pages, with code snippets to run on a nearby computer; their pages are full of examples, pictures, and fun. Code examples might include the occasional "Hello, World!" reference here or there, but they definitely involve much more than just copying, compiling and running. There is emotion.

(Of course, not everybody liked the idea. I will not link to those reviews in these pages; sadly enough, they are quite easy to find. If human nature is to be considered, such negativity against the very idea of these books was to be expected, even if the levels reached are baffling in shortsightedness and abundant in borderline stupidity. But I digress.)

As an early example of this new approach to technical books, it would be a mistake not to acknowledge the fantastic "Oh! Pascal!"[15] book series, authored by Doug Cooper from the mid-80s to the mid-90s, and which were largely responsible in the raise of popularity of the Pascal programming language during the early days of the PC.

In spite of such an illustrious predecessor, it was Kathy Sierra who triggered a major, deeper change in the way programming was taught. To be honest, the books by themselves would already have been a major triumph. Her work happened at a time where the dot-com boom[16] opened the door for new ideas, right in the middle of the Web 2.0[17] craze, and right before the rise of the smartphone and social media.

Through the use of comic images, unusual text layout, and a fantastic sense of humor, Kathy Sierra argues, readers can learn the concepts easier; tricking the brain into the proper levels of

[7] https://www.pearson.com/us/higher-education/product/Kochan-Programming-in-Objective-C-2-0-2nd-Edition/9780321566157.html

[8] https://pragprog.com/book/jaerlang2/programming-erlang

[9] http://shop.oreilly.com/product/0636920026877.do

[10] http://shop.oreilly.com/product/0636920033707.do?cmp=af-code-books-video-product_cj_0636920033707_7708709

[11] https://www.pearson.com/us/higher-education/program/Bloch-Effective-Java-3rd-Edition/PGM1763855.html

[12] http://www.thebillwagner.com/Resources/EffectiveCS

[13] https://www.oreilly.com/library/view/head-first-java/0596009208/

[14] http://shop.oreilly.com/product/9780596007126.do

[15] https://openlibrary.org/works/OL3813065W/Oh!_Pascal!

[16] https://en.wikipedia.org/wiki/Dot-com_bubble

[17] https://en.wikipedia.org/wiki/Web_2.0

dopamine. The level of self-derision is such that the cover image on the introduction chapter of "Head First Design Patterns" features a 1950s couple saying "I can't believe they put *that* in a design patterns book!"

Riding on top of the Head First success, O'Reilly started new series, most notably the "Beautiful" series, featuring books with interviews; comes to mind the fantastic "Masterminds of Programming,"[18] which we will cover separately in a future edition. Many more Head First books came out, some of which, in the cover photo, sit proudly in my bookshelf.

The Head First books are a hallmark, reaching a perfect equilibrium between accessibility and relevance; something that both the "For Dummies" series and most Addison Wesley books overshot, in absolutely and completely opposite directions. Funny without being quirky; memorable without being childish; and correct, without being pedantic.

On the other hand, they also triggered a horrid revelation, the darkest nature of online harassment, with abject levels of abuse and misogyny. Maybe they played a minor role as catalysts opening the door to the decade of the Me Too movement[19].

Cover photo by the author.

[18] http://shop.oreilly.com/product/9780596515171.do
[19] https://en.wikipedia.org/wiki/Me_Too_movement