Issue 008: Programming History

Graham Lee

May 6th, 2019

Welcome to the eighth issue of *De Programmatica Ipsum*, dedicated to the subject of *Programming History*. In this edition:

- Fernando Rodríguez[1] explains how the old is always the new[2], and how quickly it is forgotten.
- Adrian shares a short list of books[3] to learn more about the history of programming.
- In this issue's subscriber-only article, Graham delves into the complex issue of how history is written[4] – and by whom.

Enjoy this issue! Please let us know if you have any feedback[5] and get our free newsletter[6] to stay updated about new releases. If you want to support us, subscribe[7] for a month or a year, and let us know if you would like to write with us[8].

Cover photo by Nicolagypsicola[9] on Unsplash[10].

[1] https://deprogrammaticaipsum.com/user/frr149/
[2] https://deprogrammaticaipsum.com/a-brief-and-painful-history-of-programming/
[3] https://deprogrammaticaipsum.com/history-repeating/
[4] https://deprogrammaticaipsum.com/history-is-a-fiction/
[5] https://deprogrammaticaipsum.com/feedback/
[6] https://deprogrammaticaipsum.com/newsletter/
[7] https://deprogrammaticaipsum.com/subscribe/
[8] https://deprogrammaticaipsum.com/write-with-us/
[9] https://unsplash.com/photos/tW_-BwcVzM0?utm_source=unsplash&utm_medium=referral&utm_content=creditCopyText
[10] https://unsplash.com/search/photos/history?utm_source=unsplash&utm_medium=referral&utm_content=creditCopyText

A Brief And Painful History Of Programming

Fernando Rodriguez

May 6th, 2019

Kent Beck is deservedly famous for creating Test-Driven Development (TDD.) According to himself, he did not invent it. He only rediscovered it.

Beck found about this forgotten TDD technique while chatting with an older programmer. The elder explained how things were done back in the good ol' days of punch cards:

```
10: Grab a punchcard, and manually punch the output you need.
20: Start coding. Every time the mainframe spits a punchcard, compare it to the one you manually punched.
30: If both cards match, GOTO home. Otherwise, GOTO 20.
```

The elder sage was baffled to know that this was not a common practice anymore:

> Then how on Earth do you know that your code is correct?

Great question, grandpa!

Alchemists, At Best

There is a surprising amount of ideas and concepts that have been forgotten and rediscovered along our short history. This is a clear telltale that even though Computer Science is a science, programming is not. Neither is it a form of engineering. We are neither scientists nor engineers, alchemists at best.

The history of programming resembles the history of science until the Renaissance. A painful process of back and forth where each individual had no giants on whose shoulders to stand, or had forgotten them.

There's a plethora of breakthroughs that disappeared into the sands:

- Hero of Alexandria built steam engines and wind-powered machinery 2000 years before the industrial revolution.
- Aristarchus of Samos first suggested Heliocentrism in the 3rd century BC.

And the list sadly goes on.

Our Own Heroes

We have our fair share of programming Heroes and Aristarchuses. Lisp is perhaps the most obvious one, but there are many others. To name an few:

Daniel McCracken mentions TDD using punchcards in his book "Digital Computer Programming[1]", in 1957. As of today, more than half a century later, it's still not a standard practice in the average software shop.

Edward Ashroft and others first described the data-flow programming paradigm[2]. They designed an entire programming language around this concept, called Lucid in the early 80's. Only now these ideas are being rediscovered as a better solution for data-intensive apps. Jet.com is now using workflows of data[3] as a better alternative to microservices.

Ralph Griswold

Perhaps the most undeservedly forgotten of these Heroes was Ralph Griswold. At the University of Arizona, in the early 70's, he created two amazing languages: Snobol and Icon.

Learning about Snobol is like finding the remains of an ultra-advanced alien civilization while digging in your backyard.

Back in the 70's, when there were people walking around the Moon and you could buy a ticket for a supersonic commercial jet, you also had a language with delayed evaluation, pattern matching and many features not yet found in "modern" languages.

One of the key insights Griswold and his team had, was deceptively simple:

> When a function is called, only two things may happen: It either succeeds or fails.

Every function in Snobol or Icon either fails or succeeds. This information is returned for every function and is orthogonal to the return value. Having success and failure as first-class citizens of the language greatly simplifies error handling. It also allows for very flexible control-flow and even concurrency via coroutines. Only now (forty years later!) are some of those concepts entering the mainstream via the Result type[4] that the "modern" kids on the block have or coroutines (very few have).

This evolutionary process, slow and painful, where information is lost, hopefully to be rediscovered later, could be described in terms of the Myth of Sysiphus. However, Philip Greenspun did a much better job at describing it:

> Our industry is a constant source of embarrassment.

[1] https://www.amazon.com/Digital-Computer-Programming-McCracken-1957-12-03/dp/B01F9QQF8U/ref=sr_1_fkmrnull_1?keywords=Digital+Computer+Programming+mccracken&qid=1556551397&s=gateway&sr=8-1-fkmrnull

[2] https://en.wikipedia.org/wiki/Dataflow_programming

[3] https://medium.com/jettech/microservices-to-workflows-expressing-business-flows-using-an-f-dsl-d2e74e6d6d5e

[4] https://en.wikipedia.org/wiki/Result_type

What Is Dead May Never Die

If an elegant idea is carried away by the hype tide du jour, fear not. If this has been coded before, it will be coded again. So say we all!

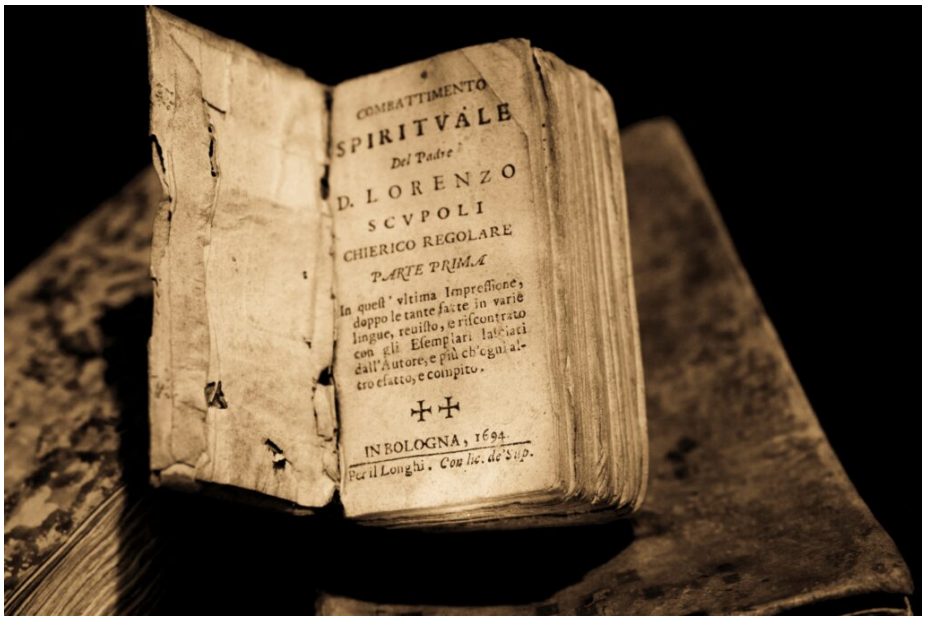
To the memory of Ralph E. Griswold (1934 -2006.)

Cover image by the author: Aeolipile, a steam engine from 2000 BC.

History Repeating

Adrian Kosmaczewski

May 6th, 2019



Somehow we all agree about the importance of history in our society. We teach it to our younger ones, we quote it in our speeches, we talk about it during our dinners. Maybe it is because we had to memorize the names of battles fought ages ago, and we expect to capitalize on that fact so as to appear wise (if not arrogant) to others. We might even agree with history itself, shaking our heads in dismay to some extent, as we see the events unfolding nowadays. A phenomenon which, by all measures, tends to increase with age.

There is an old joke circulating in Twitter[1] stating that

> Those who do not study history are doomed to repeat it, yet those who _do_ study history are doomed to stand by helplessly while everyone repeats it.

Another one[2] states, quite sadly, that

> It seems like a large portion of being over 40 in tech involves listening to younger people with Big Ideas, sighing deeply, and muttering "yes, we all knew about that 20 years ago."

Is there a way to way to avoid repeating mistakes[3] in software engineering by applying history? There is such a fast pace in this industry that by the age of 35 most software developers have a "Senior" title, and, as Steven Sinofsky pointed out[4], a rather large collection of t-shirts bearing the names of obsolete technologies.

[1] https://twitter.com/EZuelow/status/1097665212591886337
[2] https://twitter.com/lemay/status/1085215551247413249
[3] https://twitter.com/slava_pestov/status/1015366481717047297
[4] https://twitter.com/stevesi/status/998452578206736384

The study of programming history might not be the solution to all of the problems in our industry, for sure. It is also worth pointing out that most university curricula simply *do not include* any mention whatsoever of such subjects. Maybe it is time to start providing such information to students. The author of these lines certainly would approve a more general spread of this kind of information in the future.

A Short List Of Books

The following list provides a necessarily short, incomplete list of books that the author can recommend, to start an exploration in the subject of programming history.

Dealers Of Lightning

I would start with this title[5] published in 2000 by Michael Hiltzik. The book traces the history of the Xerox Palo Alto Research Center (PARC) in the seventies, arguably a defining moment for the current period of technology we are living today.

In one of the most significant parts of the book, the author described how around 1972, there was an office somewhere in California where people were using a fully functioning GUI with a mouse, writing documents on a WYSIWYG word processing applications, printing those documents on a laser printer, and even sending them via e-mail. A place, by all means, 20 or 25 years ahead of its time, and where many ideas, including that of teaching programming to kids, became reality, even though it was not the reality that Xerox wanted to build.

The Dawn Of Software Engineering – From Turing To Dijkstra

This one is a rather obscure title[6] published by Edgar G. Daylight. It is, however, an overlooked yet absolute masterpiece of incredible detail.

The author finds a quite interesting thread that links the influences of these two major pioneers of our field; through papers and references, their opinions shaped our programming languages and permeate until today in most of our day-to-day tasks.

John Von Neumann And The Origins Of Modern Computing

There are many biographies of John Von Neumann; but this one[7] benefits from the experience of William Aspray, MIT researcher with lots of experience in the subject. The book delves into all of the contributions to science and technology of Von Neumann; they range from physics to mathematics, and of course, to programming, through the now ubiquitous hardware architecture that bears his name.

The result is a book whose lecture can only confirm the reader in the belief that Von Neumann was, together with Einstein and Gödel, one of the greatest minds of the twentieth century – and to top it off, the three shared offices in the same location, the Institute for Advanced Studies at Princeton. Only the Platonic Ἀκαδημία[8] or the Solvay Conferences of Physics[9] might have brought such minds together in human history.

[5] http://michaelhiltzik.com/dealers-of-lightning/
[6] https://dijkstrascry.com/dawn
[7] https://mitpress.mit.edu/books/john-von-neumann-and-origins-modern-computing
[8] https://en.wikipedia.org/wiki/Platonic_Academy
[9] https://en.wikipedia.org/wiki/Solvay_Conference

Programming Languages: History And Fundamentals

The last one in the list, but by all means the greatest, is the summary by Jean E. Sammet[10], published in 1969 by Prentice-Hall and nowadays out of print, of the state of programming languages before Neil Armstrong set foot on the moon.

Outstanding in breadth and depth, the book covers around 120 programming languages available at the time of its publication, through 800 bibliographic notes. Some of those languages are still relevant today, like COBOL and FORTRAN, while most others have been left behind by technology and human memory. Each of those languages is studied in the context of their creation; the reason behind their existence, the hardware they required, and the teams involved in their creation. Archive.org contains the scanned promotion leaflet of the book[11] worth a look.

Conclusion

There you go; four books, roughly ordered by their reading complexity, each providing the same conclusion in the mind of the reader: those wise words of Dame Shirley Bassey in that classic song[12] of the nineties:

> The newspapers shout a new style is growing,
> But it don't know if it's coming or going,
> There is fashion, there is fad;
> Some is good, some is bad;
> And the joke is rather sad,
> That it's all just a little bit of history repeating.

Cover photo by Daniele Levis Pelusi[13] on Unsplash[14].

[10] https://www.amazon.com/Programming-Languages-Fundamentals-Automatic-Computation/dp/0137299885

[11] https://archive.org/details/TNM_Programming_Languages_history_and_fundamental_20171115_0058

[12] https://www.youtube.com/watch?v=yzLT6_TQmq8

[13] https://unsplash.com/photos/EHfiW-cV0ls?utm_source=unsplash&utm_medium=referral&utm_content=creditCopyText

[14] https://unsplash.com/search/photos/history-repeating?utm_source=unsplash&utm_medium=referral&utm_content=creditCopyText

History Is A Fiction

Graham Lee

May 6th, 2019

What is the history of programming? There is none. There are many. It is too long. Let me summarise.

There is no history of programming, in the sense of a recording of the events and developments related to the field. Many of these events went unrecorded, or ignored, or lost. Even the important ones.

Two significant early deployments of digital computers were the Colossus, designed and built in the UK, and the ENIAC in the US. Both were military projects. We may know something of these machines – in the case of Colossus, enough to build a replica – but not their history.

Many programmers like to lead a review with the question: what else did you try? So what trade offs led to the design of the Colossus? How did the maintenance budget affect the choice of technology? Was the lifetime of the project considered? What else could ENIAC have been? What else did you try?

History Is Written By The Authors

In addition to lacking enough history, we suffer from too much. Anyone with a writing implement and the ability to order events chronologically – correctly or otherwise – can construct a narrative where none exists.

A great way to understand the capriciousness of history is to read business nonfiction. Go to your local library, and take out a few books by American authors on succeeding at business. They will have gold embossed titles like romance fiction, and proudly tell you that you can "skip the MBA" if you read this one book.

Look up either Steve Jobs or Howard Schultz in the index. They'll be there. Read what each author has to say. It will tell you that Apple, or Starbucks, succeeded because the CEO did the thing that is the title of the book in embossed gold letters.

Now you'll need a second hand bookstore or a bigger library for the next step. Find business books from the 1990s. Look up whichever revolving-door Apple CEO was in seat at the time. They'll be there. This time, the author tells you that Apple is failing because they did not do the thing that is the title of the book.

We find that the narrative is made to fit the events, and select events at that. History is written by the people who write history, not by the people who make it. Because there is no "making history". There is only selection and interpretation.

Homo Economicus

History is an attempt to tell the story of why some thing turned out the way it did by showing how events led up to that thing. But the people involved at the time didn't necessarily know about those other events. Or they didn't care. Or they knew about other events, which historians are ignorant of.

In this way, history can be like economics. Economists say that people or businesses act in the ways that they do because of bounded rationality. In fact, both the bounds and the rationality were invented by economists. People act as they do because of emotion, and whim, and caprice. Ask somebody why they made a particular choice, and they may be able to construct a *post hoc* rationalisation, but they are probably guessing.

History is just economic *post hoc* rationalising on the scale of a society. Somebody wants to tell a story about why the world is the way it is, so they look for records of events and fit them into their story. Like Rudyard Kipling, they want to explain why things are "just so".

On Programming

All of this is not to destroy programming history. I mean simply to cast it in the light it deserves. A light under which we apply a healthy dose of scepticism.

A summary history of programming by an Agile consultant is likely to talk up the Snowbird meeting and successful Agile projects. It might be silent on successful clean-room engineering projects undertaken by IBM.

The history written by a software engineering academic may talk up particular journal articles or conference proceedings, ignoring that many systems have been without knowledge of them. Meanwhile, a "pragmatic", "move fast and break things" historian might downplay the relevance of stuffy academic journals. That author is still likely to drop the occasional "considered harmful" or "premature optimisation" into conversation.

A Linux fan will tell you that Linux was at the "right place, right time" when post dot-bomb companies needed to renew their proprietary Unix licenses without owning much cash. They may not mention that FreeBSD, OpenBSD, NetBSD, Minix and others were in that place at that time, too.

Someone who wants to explain how Apple introduced graphical computing to the mass market will talk about Steve Jobs taking the GUI from Xerox PARC, like Prometheus taking fire from Mount Olympus. They might not mention that the first GUI to achieve mass market success was actually Windows 3.1.

History Is Live

All of this is to say that programmers should not consume histories of programming. Not uncritically, anyway. We *are* its history. Choose values, principles and practices because they work for you, not because they "won".

They did not "win". There was no competition. There may have been competition between businesses, but not between technologies or tools. Certainly not between values, principles, and practices. Besides, businesses "win" by remaining in business, not by having better values or superior technology.

Cover photo by Giammarco Boscaro[1] on Unsplash[2].

[1] https://unsplash.com/photos/zeH-ljawHtg?utm_source=unsplash&utm_medium=referral&utm_content=creditCopyText

[2] https://unsplash.com/search/photos/history?utm_source=unsplash&utm_medium=referral&utm_content=creditCopyText