De Programmatica *Ipsum*

DE PROGRAMMATICA IPSUM

Issue 098: Apple

November 2nd, 2026

Table of Contents

Issue 098: Apple 5
How Could That Possibly Work? 9
The Subtle Art Of Hating Developers 15
Steve Gets Trolled 21
Sonia Devillers 25
Jim Carlton 31
Andy Hertzfeld 35

Issue 098: Apple

November 2nd, 2026

Welcome to the 98th issue of *De Programmatica Ipsum*, about *Apple*.

In this edition:

- Graham reminisces on his own experience as a software engineer at Apple and its peculiar methodology choices[1].
- Adrian points out Apple's infamous disdain[2] of third-party developers.
- In our Vidéothèque section[3], we watch Steve Jobs getting trolled[4] during WWDC'97, and then French journalist Sonia Devillers[5] analyzing the terrible "Crush" advertising campaign.

- In the Library section[6], we review "Apple: The Inside Story of Intrigue, Egomania, and Business Blunders" by Jim Carlton[7], and "Revolution in the Valley: The Insanely Great Story of How the Mac Was Made," by Andy Hertzfeld[8].

Download this issue in DRM-free PDF[9] or EPUB[10] format, and read it on your preferred device. You can also subscribe to our RSS feed[11], featuring the full content of our articles.

We would like to thank our patrons who generously contribute every month (or have contributed in the past) to our work and help us run this magazine. Thank you so much! In alphabetical order: Adam Guest, Adrian Tineo Cabello, Benjamin Sheldon, Christopher Nascone, Colin Powell, Franz Lucien Moersdorf, Guillermo Ramos Álvarez, Jean-Paul de Vooght, Dr. Juande Santander-Vela, Patryk Matuszewski, Paul Hudson, Quico Moya, Roger Turner, Szymon Licau, and countless more leaving anonymous tips every month.

Enjoy this issue! Please share our articles on social media, or contribute[12] if you would like to support our work with a donation via Liberapay[13].

Cover photo by Zhiyue[14] on Unsplash[15].

References

[1] https://deprogrammaticaipsum.com/how-could-that-possibly-work/
[2] https://deprogrammaticaipsum.com/the-subtle-art-of-hating-developers/
[3] https://deprogrammaticaipsum.com/category/videotheque/
[4] https://deprogrammaticaipsum.com/steve-gets-trolled/
[5] https://deprogrammaticaipsum.com/sonia-devillers/
[6] https://deprogrammaticaipsum.com/category/library/
[7] https://deprogrammaticaipsum.com/jim-carlton/
[8] https://deprogrammaticaipsum.com/andy-hertzfeld/
[9] https://deprogrammaticaipsum.com/pdf/issue-098-apple.pdf
[10] https://deprogrammaticaipsum.com/epub/issue-098-apple.epub
[11] https://deprogrammaticaipsum.com/index.xml
[12] https://deprogrammaticaipsum.com/contribute/
[13] https://liberapay.com/akosma/donate
[14] https://unsplash.com/@zhiyue?utm_source=unsplash&utm_medium=referral&utm_content=creditCopyText
[15] https://unsplash.com/photos/silhouette-of-people-standing-near-wall-7DOU5NlNIcE?utm_source=unsplash&utm_medium=referral&utm_content=creditCopyText

How Could That Possibly Work?

By Graham Lee

I have got a pretty clear model of how software engineering works, or is supposed to work. Like most people, I am a product of my time and my experiences, with new information and experiences getting incorporated (or sometimes filtered) through the weight of my history. As someone who became a professional software engineer at the start of the 21st century, those experiences mostly come from the period when object-oriented programming hit the plateau of productivity[1] and agile software development rode the whole hype cycle, with some teams now

being productive and many more mired in the trough of disillusionment through botched or misunderstood adoptions.

In many ways, this period coincides with the death of software engineering methodology. Methodology is the study of method; not the engineering of software, but the engineering of software engineering. The people who went on that famous Snowbird skiing holiday that generated the Agile manifesto were a mix of people who make software, people who manage people who make software, people who tell other people how to make software, and people who think about how people go about making software; with that last group being the methodologists. These people compared notes on the ways in which they worked, which were all biased towards so-called "lightweight methods" that optimise for learning what people want, making software that does that, then checking whether anything's changed. What they came up with was a document that said to do that. Get software out to your customer as quickly as you can, work with them to understand what to do next, and do not be afraid of finding out that they (now) want something else.

That is such broadly applicable (and vague) advice that it has never been superseded. Everyone thinks they are doing that, whether they are following by-the-book Scrum, a poorly understood Scrumbut process, or any of the other Lean, XP, Kanban etc. processes that you come across. The only real attempts at modernising methodology in the quarter-decade since Agile killed it have been devops and continuous delivery, and those have both been restatements of the ideas of agile software development. The core message of devops—that everyone working on software should work on the same thing—is the message about "self-organising teams" that is one of the principles behind agile software development. Continuous delivery is even a phrase borrowed from that same list of principles.

Instead of improvements to methodology, we have tool advocacy—the idea that a new programming language, large language model, or cloud deployment platform is a silver bullet that solves every problem. With no new approaches to software design since OOP, new programming languages offer "post-object" objects that are basically the same thing but with slightly different approaches to memory management or threading. Design itself has embraced the JFDI—Just Fucking

Do It—ideology, with design choices being laundry lists of technologies to adopt where the choices are explicit at all.

That means that when I walk into a new software shop—or connect to their group video call, which is much more likely these days—I have a good idea of what to expect. There is going to be a two-pizza team, probably mostly programmers with an "ops person" or two, a "QA person" or two, and a "product owner". When they describe their architecture, it is actually their technology laundry list, which is mostly open source components with maybe a couple of proprietary bits, commonly Slack, Github, AWS/GCloud/Azure, and perhaps the commercial version of something like MongoDB. They have a long "backlog" of tickets in a task tracker or a virtual kanban board, and once every 1-4 weeks the product owner suggests which ones to work on next. The team work out how many they think they can do, take those away, assign each of them to a programmer, and then when each programmer is done they assign their ticket to the QA, and then it goes to the ops person, and then it is in "staging" or "pre-prod", and so on.

I also already have a good idea of why that team is getting slower at delivering on expectations from "the business", why customers are becoming less satisfied with the product, and why a lot of the people are grumpy; and a few solid interventions to turn that around and create a high-performing team. You will find my rates very reasonable, by the way[2].

While that is my model of how software teams *do* work and *should* work, not every team that I join *does* work like that. And some of the surprises are eye-opening. Take Apple, a company that I have worked for as both a contractor and employee, across multiple cross-functional teams led by sponsors in different divisions. You count up the number of people employed as software engineers and related roles and you are well into five figures. They report to multiple different sub-organizations in the company, with the effect that sometimes the first manager in common between two programmers who need to work together might be the CEO.

You will not find much information about how Apple works, beyond this HBR article[3] that they are happy to share on their website. What it describes is a functional, not geographic or sectoral, division of labour. Each engineer is in a team responsible for a different part of a huge, integrated system, with thousands

of components that come together to make about half a dozen operating systems, productivity apps, creativity apps, server components, retail support systems, and so on—products that involve thousands of engineers where any pair might not directly communicate at all during the course of a release. How can that possibly work?

Well, the answer appears to be "very nicely, thank you very much". They routinely have successful product launches, and a higher revenue or profit per employee than any of those two-pizza teams that are a better fit to my mental model. Fred Brooks and Melvin Conway tell us that communication problems should overwhelm productivity as organizations get larger, and yet here is one of the largest software orgs on the planet also carrying on being one of the most productive. Sure, there are bugs, and people who do not like the products, but that is largely the result of trying to please billions of people with the same products. The list of things that literally do not work, as a fraction of the list of things that *should* work, is not all that big.

And that is really the lesson I want to communicate when it comes to software engineering methodology: one that is more about you and me than it is about Apple. Any organization that is well outside your comfort zone is not necessarily doing it wrong, they are doing it differently. If my experience over the last 20 years had mostly been at Silicon Valley giants, then I would have a model of software engineering that matches what they do, and I would be very shocked by those little two-pizza teams who try to run a whole business from a single room. Approach new situations with humility, and an eagerness to adapt or discard your model. Of course look for opportunities to improve the system, but also pay attention to the things that do not need improving just because they do not align with your preconceptions. You might find that they work very nicely, thank you very much.

Cover photo by Shelley Pauls[4] on Unsplash[5].

References

[1] https://deprogrammaticaipsum.com/the-hype-cycle-of-oop/
[2] https://sicpers.info/agency
[3] https://www.apple.com/careers/pdf/HBR_How_Apple_Is_Organized_For_Innovation-4.pdf
[4] https://unsplash.com/@shelleypauls?utm_source=unsplash&utm_medium=referral&utm_content=creditCopyText
[5] https://unsplash.com/photos/red-apple-fruit-I58f47LRQYM?utm_source=unsplash&utm_medium=referral&utm_content=creditCopyText

The Subtle Art Of Hating Developers

By Adrian Kosmaczewski

For a short period in time, around the end of the 1990s and beginning of the 2000s, it seemed as if Apple actually cared about third-party developers hard at work creating software for their platforms. Of course this interest waned as soon as it was clear that Apple was on track to become the most valuable company on the face of Earth. Getting software developers interested in Apple was always a one-way road paved with obstacles.

During those weird times, around the turn of the millennium, Apple was desperately trying not to go bankrupt and forgotten. So Steve Jobs did something that he really hated doing: opening their platform for outside developers to create software for it.

In those long-forgotten days, Apple would have a strong commitment to open sourcing as many software thingies from their ecosystem as possible and touting that through a now largely forgotten web page[1] nobody cares about. It also made sure Java developers felt at home[2], even allowing them to create Cocoa applications with it, instead of having to use the (ugh!) Objective-C programming language.

But there were clear limits to that unusual love for developers: for example, the iPod, which is celebrating its 25th anniversary[3] as this article hits the web, was famously closed for external developers, and never saw anything resembling an App Store for it. And to be honest, even the iPhone was about to suffer the same fate when it was first released in August 2007; we had to wait until March 2008 to have the confirmation of an upcoming "iPhone SDK," which turned out to be, ironically enough, quite the cash cow for the company.

Because it is a quite accepted fact by all biographers of Apple and Steve Jobs that the man *hated* third-party software developers with passion. The man did not want to see software he had not personally approved on those machines he was selling.

It was not always like this. Here is a famous example: in page 34 of the May 1977 issue of BYTE Magazine[4], Steve Wozniak himself proceeded to describe in astonishing detail the ins and outs of the Apple II, arguably one of the most popular 8-bit computers of all time. His description included various snippets of sample code, board electronics schemas, and various screenshots.

> *To me, a personal computer should be small, reliable, convenient to use and inexpensive.*
> *(...)*
> *The latest result of my design activities is the Apple-II which is the main subject of this system description article. The Apple-II builds upon this idea by providing a computer with more memory capability, a read only memory (ROM) BASIC*

interpreter, color video graphics as well as point graphics and character graphics, and extended systems software.

Imagine Apple providing such circuit schemas nowadays, for example for their latest MacBook Pro with PROLED XDR touchscreen and M5 Max CPUs, enabling the Asahi Linux[5] team to finally deliver a proper experience with full driver support. The dream!

But… no. The issue is that the idea of offering an appliance to consumers (a mantra that pretty much defines Apple in its entirety nowadays) is anathema to that of opening their (admittedly crippled) APIs to others to play with, no matter how hard said developers scream.

This attitude is (maybe) also what explains why Apple makes arguably the best computers on the market: fast, lightweight, snappy, and (relatively) secure. But also crippled by what increasingly appears as a bloated operating system, filled with a gazillion features hastily copied from others, exposing an array of quite broken APIs to software developers, and offering a set of programming tools to third-party developers that cannot be called anything else but pathetic.

These days, the only reason a software developer should choose a Mac as a software development tool is, precisely, to build software for the many Apple official operating systems (macOS, iOS, iPadOS, watchOS, tvOS, and whatever runs CarPlay or the recently released HomePod). The rest of the world could not care less about those restrictive APIs, or about the sad programming language (Swift) with which those developers should be fighting all day long to get shit done.

While the rest of the world moves into automation, CI/CD, DevOps, containers, Kubernetes, AI, and whatnot, developers stuck in the Apple ecosystem must still fight against Xcode and its various shenanigans to get anything done. The situation has not improved in a whole decade[6], which clearly shows that not giving a shit about developers is just about corporate policy at this point.

(Yes, this author knows that Apple has provided an implementation of the OCI standard[7] in the form of the open-source container project on GitHub[8], but to be honest, I still have to see one of those containers in the wild at this point.)

Apple's legendary developer micromanagement chromosome has always been problematic, but right now it is ridiculous. Of course, they now own the complete software delivery platform, from the CPUs to the IDEs and the marketplaces, so it is ridiculous to expect any betterment in the developer experience anytime soon.

Sadly, the only way the company could react to this state of things would be a major liquidity and bankruptcy event, but Apple's cash reserves (unironically generated in great proportion through the hard of indie developers) have never been so humongous at any other time in history, so the recommendation of the author of these lines to any person new to IT is to avoid dealing with Apple-sanctioned developer tools as much as possible.

Cover photo by John Warg[9] on Unsplash[10].

References

[1] https://web.archive.org/web/20020126133130/https://opensource.apple.com/
[2] https://web.archive.org/web/20011107065120/http://developer.apple.com/java/index.html
[3] https://sixcolors.com/post/2026/10/the-ipod-which-set-apple-on-its-course-turns-25/
[4] https://archive.org/details/BYTE-MAGAZINE-COMPLETE/197705_Byte_Magazine_Vol_02-05_Interfacing/page/34/mode/2up
[5] https://asahilinux.org/
[6] https://akos.ma/blog/courage/
[7] https://opencontainers.org/
[8] https://github.com/apple/container
[9] https://unsplash.com/@specphotops?utm_source=unsplash&utm_medium=referral&utm_content=creditCopyText
[10] https://unsplash.com/photos/primitage-road-no-warning-signs-signage-V12_AmFAINg?utm_source=unsplash&utm_medium=referral&utm_content=creditCopyText

Steve Gets Trolled

By Graham Lee

You have almost certainly seen the video before[1], because it gets trotted out on LinkedIn every time someone wants to boast about how humble they are. The context: Apple's Worldwide Developer Conference (WWDC), 1997. The dramatis personae: an alleged[2] Robert Hamisch, allegedly an engineer at a Sun Microsystems reseller, and Steven P. Jobs (verified, blue checkmark), CEO of Pixar, Inc. and advisor to Apple Computer, Inc. after the purchase of his computing startup, NeXT, Inc.

Maybe-Hamisch opens (in response to an earlier question about the demise of OpenDoc[3], a technology for building componentised document-centric software instead of monolithic applications) with "more of a comment than a question":

> *Mr. Jobs, you're a bright and influential man. It's sad and clear that on several counts you've discussed you don't know what you're talking about. I would like for example for you to express in clear terms how say Java in any of its incarnations addresses any of the ideas embodied in OpenDoc. And when you're finished with that perhaps you could tell us what you personally have been doing for the last seven years.*

This person could have merely waited 29 years to grab a copy of Steve Jobs in Exile[4] and discovered that the answer to the last part is that Steve was raising a kid, running Pixar, and putting in sporadic levels of effort when it came to running NeXT, hyper-focusing on some details while leaving others untouched or even sabotaging parts of the work.

However, he stated the not-question, and the rest is history: Steve pauses, drinks his water, pauses some more, then gives an answer. An answer that simultaneously demolishes the request while also allowing for the possibility that it might be true. That Apple's responsibility is to start with the customer experience and work back to the technology they need, not to build cool technology and try to work out how to market it. Some mistakes will get made on the way, "and that's good because it means decisions are being made".

And it is true; both that Apple became an experience-focused company that first gave us computers that do not crash when you so much as look at them, then as many songs as we could fit in our pockets, then real pocket web browsers. And also that they made mistakes along the way.

Let us go back to the original almost-question, which asked in what ways *Java* was comparable to OpenDoc. At the time, Apple were huge Java fanboys, and Gil Amelio (then CEO of Apple Computer) reports discussing Java and Rhapsody with Steve in their advisory meetings.

When you run a "beleagured" computer company with a tiny market share, Java offers clear benefits: any software someone writes in Java automatically becomes available on your platform. There are no killer apps[5] for your computer any more, but then there are not any for your competitors' computers either; everything runs everywhere. But more than that, Java offered the first scaled-up realisation of the object-oriented vision.

Apple had bet on Java with the Mac OS Runtime for Java[6] (MRJ), which supported Java software and browser applets on the classic Mac OS. When they integrated the NeXT technology, they went even deeper, providing Cocoa-Java (née Yellow Box Java) access to the OPENSTEP APIs from the Java virtual machine, and even replatforming WebObjects entirely onto Java.

Launch a developer release of Rhapsody (the transitional, server version of OPENSTEP on the Mac that became Mac OS X) or the early versions of Mac OS X and you are running Java everywhere, because someone at Apple wrote it once. At various times, the TextEdit app, System Preferences, and even Login Window got ported to Java—and then ported back to Objective-C, after people noticed that their Macs ground to a halt for a minute or so while they loaded the Java runtime environment from a slow spinning hard disk.

Further, developers did not really get onboard either. A lot of long-running Mac developers were betting on Carbon (either directly, or indirectly via PowerPlant) to keep their investment in the classic Mac OS technology valid, the refugees from NeXT knew that Objective-C was superior to Java anyway so did not feel the need to switch, and people new to the platform similarly seemed happy enough with ObjC.

So in a way, maybe-Hamisch was correct. It *was* a mistake to bet on Java. But Steve was also correct: the goal was to make things that customers want, and there are a billion more Apple customers today than there were on that day in 1997.

Cover snapshot chosen by the author.

References

[1] https://www.youtube.com/watch?v=oeqPrUmVz-o

[2] https://www.quora.com/History-of-Apple-Inc/History-of-Apple-Inc-Whos-the-guy-who-insults-Steve-Jobs-in-this-clip-of-WWDC-1997

[3] https://www.youtube.com/watch?v=_7Yz-NRBv4Q

[4] https://geoffreycain.net/steve-jobs-in-exile/

[5] https://deprogrammaticaipsum.com/behold-the-app-killers/

[6] https://www.infoworld.com/article/2169105/four-mac-based-ides-compared-which-should-you-choose.html

Sonia Devillers

By Adrian Kosmaczewski

On September 22nd, 1984, the then French president François Mitterand and West Germany chancellor Helmut Kohl visited the Douamont Ossary, paying tribute to the soldiers who fought in the infamous battle of Verdun[1] during World War I. In a highly symbolic gesture, both chiefs of state held hands[2] during the ceremony, showing the reconciliation between the French and the German, less than 40 years after the end of World War II, and merely 5 years before the fall of the Berlin Wall.

As part of this reconciliation, in 1991 started broadcasting in continental Europe a new, quite innovative binational TV channel, called "Arte"[3]; a very poetic name that apparently also serves as an acronym for "Association relative à la télévision européenne." A distinctive feature of this channel, available in most cable TV subscriptions for decades, was that it included programs produced in both French

and German languages, with subtitles or dubbing whenever required. Arte quickly gained recognition for the quality of its productions, the depth of its analysis, and the pan-European perspective it brought to otherwise segregated cultures.

A sign of the times, Arte is nowadays arguably the most interesting channel you could watch on YouTube and is also available as a specialized app in most smart TVs and all smartphones on the market. The content showcased therein is nothing short of astonishing, from documentaries to debate to music, and in the latter case, from live classical concerts to the wildest of electronica raves. It could be described as the antithesis to TikTok: a place where you can find "content" (what a horrendous word, really) that makes you think.

Remember thinking? Yeah.

Among those programs, there is this series of shorts called "Le dessous des images," a French name that translates to "Behind the Scenes" or "The Story Behind the Pictures." As is the tradition in Arte, the name of the series is also translated to German, in this case "Mit offenen Augen," or literally, "With Open Eyes." Both names complement each other and describe the idea very well: in each edition, French journalist Sonia Devillers[4] explains the rationale, the creation, and the cultural impact of a particular photography or video in our world. A simple idea at first, yet a monumental and essential work.

Madame Devillers started her career as a journalist in the French newspaper "Le Figaro" and progressively moved into radio and, finally, in 2022, started hosting "Le dessous des images" on Arte. Initially only available online, the series ended up being broadcasted on the main TV channel from 2023 onwards. In this program she dissects the ins and outs of those images that make our world: those memes that circulate online, those reels on Instagram, those photographs on press outlets. Images that flash into our brain and, in a single sight, tell a whole story about the (sometimes sad) state of our world.

Thus came to be a criticism of a particular advertising campaign from 2024 by Apple, previously available on YouTube[5], but do not click on that link! The video has been made private since and still is at the time of the publication of this article.

Apple diehards and fanbois will probably remember it well: a massive press, compressing a wide set of objects–a piano, computer screens, a record player, a typewriter, a guitar, books, a lamp, reminiscing of Pixar's "Luxo,"[6] an arcade game from the 1970s, sculptures, tables, and cans of paint–in a massive and frankly disgusting process, somehow inspired by the wonders of those ~~statistical bullshit machines~~ LLMs, falsely referred to as Artificial "Intelligence."

And at the end of the video, in a perverse act of magic, all of those things merge and combine and disappear under the weight of the press in the shape of "the new" (at the time) iPad Pro.

I do not think we need to say more.

Apple has a long and fruitful history of advertising[7] feats. Come to mind the canonical examples, like the "1984"[8] Macintosh commercial directed by Ridley Scott himself, the 1999 "Think Different"[9] campaign, or the memorable and goofy "I'm a Mac, I'm a PC"[10] series, a real hit two decades ago.

But this campaign, well, needless to say, it was a miss. And the voices online claimed blood, as is usually the case with social media. In this case, truth be told, this author agrees with their observations. This campaign was a rare event in tasteless production, and even worse, coming from a company that has usually done a better job.

The backlash was such that in a very rare instance of corporate humility and real courage[11], Apple's vice president of marketing Tor Myhren publicly apologized[12] for it:

> *We missed the mark with this video, and we're sorry.*

Apology accepted, Tor.

Some images, like that of Kohl and Mitterand holding hands, inspire a sense of history in the making, of progress, of reconciliation, leaving centuries of animosity behind them and opening the door to new possibilities. Some others, however, like in the case of Apple's infamous "Crush" advertising, feed a narrative of disdain and hubris that is unnecessary, horrid, and frankly tasteless. Such a narrative opens the

door to the dismissal of centuries of work and craft in the name of a sacrosaint "modernity" that is not.

Maybe the most important message in this episode of "Le dessous des images" for those in our industry is that the past must not be crushed, synthesized, or forgotten but studied, cherished, and learned from. A lesson that many easily forget after joining the industry: their eyes blurred by all the hype around those shiny new things surrounding us all the time.

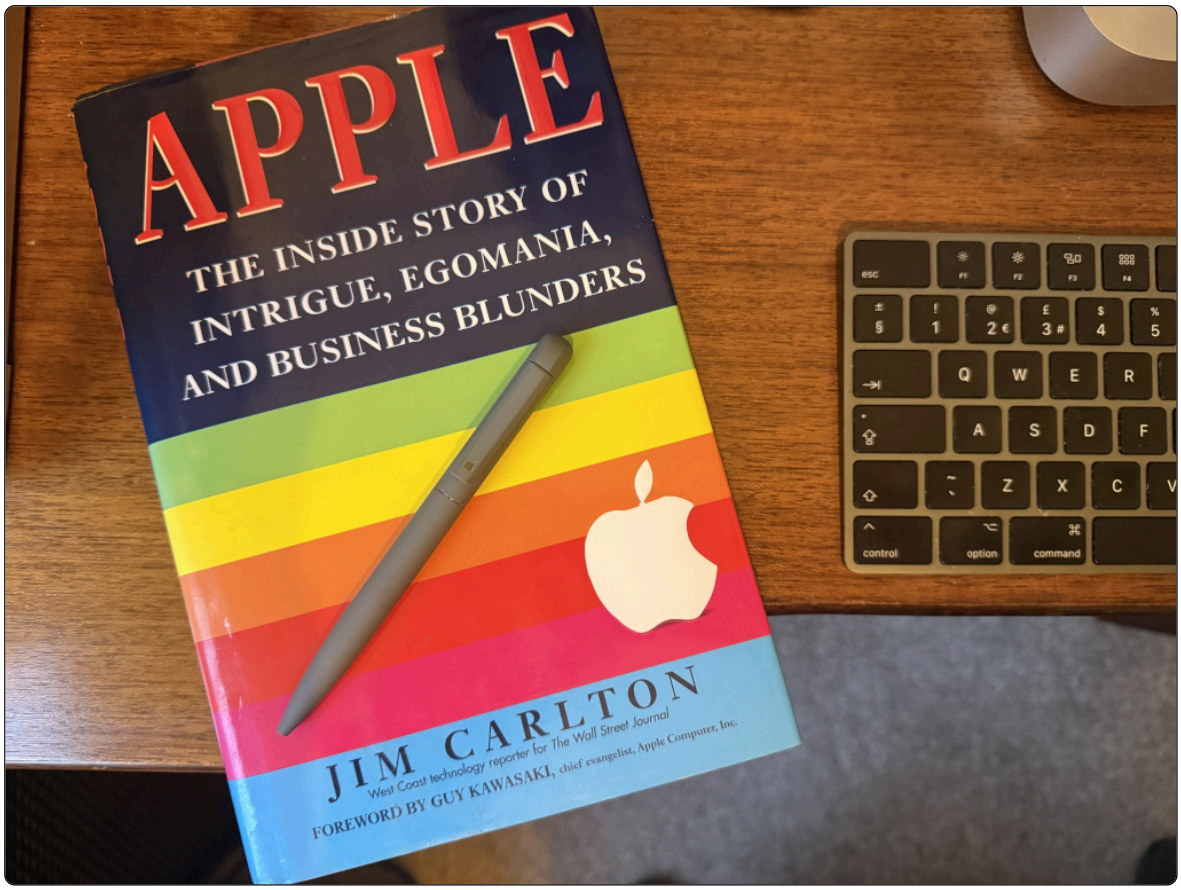
Watch "Apple: la pomme écrasée par sa pub"[13] (in French, but featuring autogenerated English subtitles) on the "Le dessous des images"[14] channel on YouTube.

Cover snapshot chosen by the author.

References

[1] https://en.wikipedia.org/wiki/Battle_of_Verdun
[2] https://en.wikipedia.org/wiki/Mitterrand_and_Kohl_holding_hands_in_Verdun
[3] https://en.wikipedia.org/wiki/Arte
[4] https://fr.wikipedia.org/wiki/Sonia_Devillers
[5] https://www.youtube.com/watch?v=ntjkwIXWtrc
[6] https://en.wikipedia.org/wiki/Luxo_Jr.
[7] https://en.wikipedia.org/wiki/Apple_Inc._advertising
[8] https://en.wikipedia.org/wiki/1984_(advertisement)
[9] https://en.wikipedia.org/wiki/Think_different
[10] https://en.wikipedia.org/wiki/Get_a_Mac
[11] https://akos.ma/blog/courage/
[12] https://www.dezeen.com/2024/05/10/apple-ipad-advert-apology/
[13] https://www.youtube.com/watch?v=7ocdOYp56tk
[14] https://www.youtube.com/@dessousdesimages01

Jim Carlton



By Graham Lee

You may be surprised to hear it, but Apple Computer, Inc. did not survive the turn of the millennium. This was the topic of the 1997 book "Apple: The Inside Story of Intrigue, Egomania, and Business Blunders" by Jim Carlton. Despite still having a couple of years to go, it was clear to Carlton that Apple was on the way out, and that his book was a fitting epitaph for "this once-great icon of American business." The only potential way for the company to survive would be a miracle.

In fairness, the history parts of the book are accurate as they go. Apple was already on a downturn by the time the Macintosh became a huge hit that turned

the company into a multi-billion firm. John Sculley, the sugar-water seller from Pepsi who had successfully turned Apple into a lifestyle brand, had countered a boardroom coup from company co-founder Steve Jobs by turning it into a "him or me" situation, and the board had chosen Sculley.

Unfortunately, it turned out that Sculley did not know as much about turning small electronics devices into desirable customer experiences as Jobs did (and, as NeXT would discover, Jobs did not know as much about it as he thought he did either). Further, while Steve had told the board that he would take a few low-level employees to go and work on his startup, he had actually poached some key people which left gaps throughout the organisation.

Apple did still have some great people, including Jean-Louis Gassée, the man who had clued the board into Jobs's machinations, but none of the people at the top could agree. Or maybe John Sculley could not get along with anyone. Regardless, Gassée was out soon, having disagreed on how to position the Mac (he thought it was a premium product that should command a premium price; others wanted to extend down into the lower end of the market, including by licensing their technology to partners).

A pattern was established; the director of European operations would move into a key role in the Cupertino HQ. But it was not Gassée who became CEO in 1993, having already been fired and started Be. Ironically, Sculley was also interested in licensing Apple's technology, and the board could not stomach it, along with his focus on one of Gassée's pet projects, the Newton; he was forced out and replaced by COO Michael Spindler, who had come from the European office.

"I never was an operations guy", said Spindler. As you can probably tell, a company that had a "not operations guy" as director of European operations, then as COO, had some problems. Those problems continued when Spindler became CEO. His great idea was a sell-off of the company. In fairness Apple and Sun Microsystems seem to have had a recurring meeting in the calendar about merging, but could never make it stick. Given that this was Spindler's only idea (the target of the merger being the variable part), he was out.

And so in came Gil Amelio, who only has the position as Apple's *second*-shortest CEO tenure because John Ternus needs to last another 438 days (from the day of this article's publication) to overtake him. Famous as the CEO who tried to point the sinking ship in the right direction, Amelio oversaw an Apple with confusing product lines, directionless engineering departments working on competing products, almost no cash, and inventory that mysteriously moved between warehouses depending on whether Apple or a reseller were preparing their accounts.

He is often derided as the worst person ever to run Apple, but only because the subsequent incumbent was very uncomplimentary. The Apple that Amelio took over had a few weeks of cash in the bank; the Apple that he left had a few months of cash in the bank, and the wherewithal to spend $400M on acquiring cutting-edge technology and expertise. However, part of that money went on hiring the CEO of the acquisition company as an advisor (Steve Jobs again), and the advice was that the CEO needed to go. Jobs himself was uninterested in running Apple, as he was having too much fun over at Pixar.

And that is where the story ends, as far as Carlton's publishers are concerned. It would be a short while before Apple appointed Jobs as the interim CEO (iCEO), but the book had already gone to the press at that point. Carlton suggests a few potential candidates for 1997's least-competitive CEO vacancy:

- Bob Frankenberg, formerly CEO of Novell.
- Eric Schmidt, then CEO of Novell, potentially most famous today as the long-time CEO of Google.
- Del Yocam, former Apple COO and CEO of Borland.

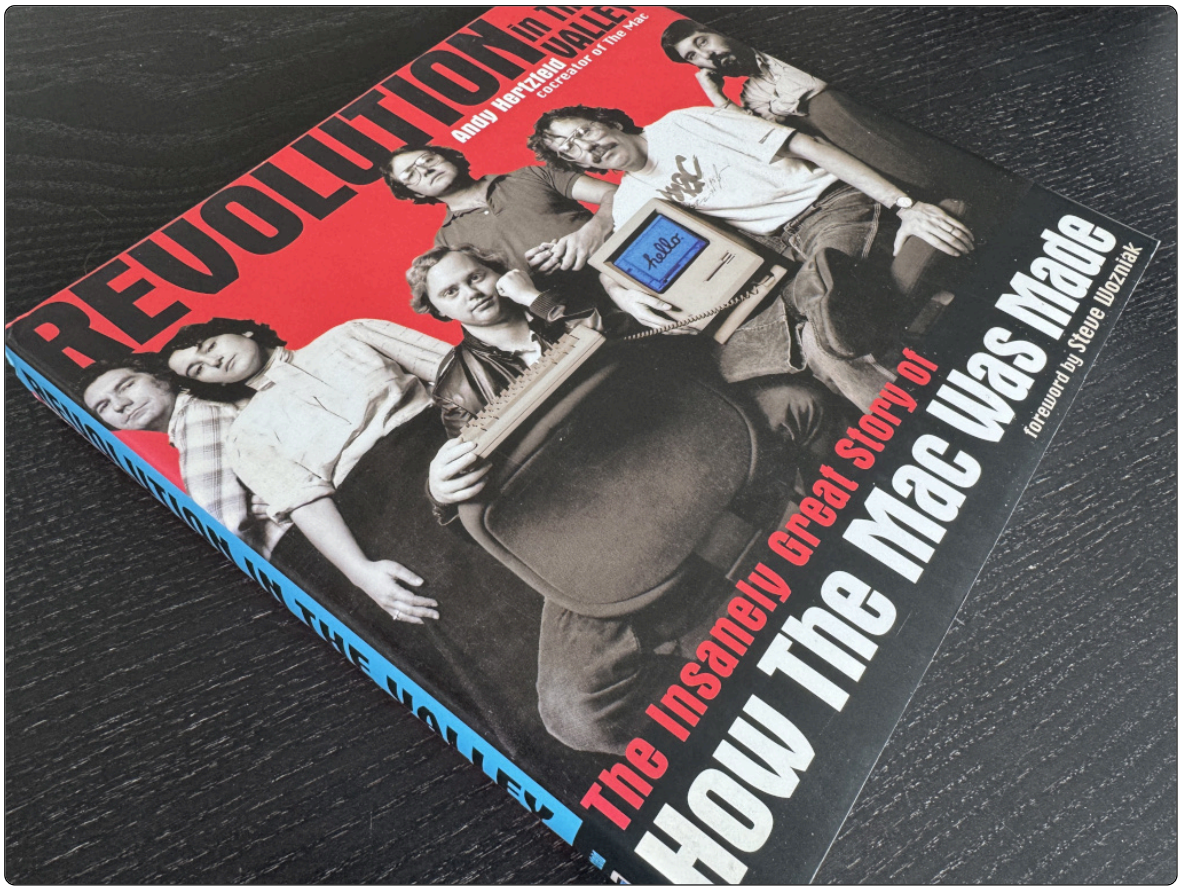
He suggests whoever takes the job has their work cut out. With Windows NT being the clear leader in the operating system market, Apple's strategy of dividing its attention between the legacy Mac OS and the Rhapsody platform based on NeXT's technology looks hugely risky. Indeed, so bad a poison apple (sorry) is the Apple platform, that even the clone makers are trying to get out of the Mac market and into Wintel machines. Microsoft is already winning accounts in Apple's key markets, education and DTP, and Intel's new MMX technology makes their CPUs competitive with the otherwise superior PowerPC architecture for graphics-rich applications.

Carlton suggests that the best thing to do with Apple is to pursue the often-abandoned merger strategy; perhaps with a consumer electronics leader like Sony or Philips, perhaps with a telco like AT&T or MCI. A telco merger would open up the opportunity to sell "smart phones, or portable phones with computer features".

Let us see how that works out.

Cover photo by the author.

Andy Hertzfeld



By Adrian Kosmaczewski

The Twentieth Anniversary Macintosh[1], released in April 1996 to commemorate the 20th anniversary of Apple, is a machine that few people remember, and with reason. It was named the 25th "Worst Tech Product of All Time" by PC World magazine, and to be honest, it was hideous and, needless to say, *insanely* expensive. Of course, as soon as Jobs returned to the company, the machine was promptly dismissed and phased out.

The word "insanely" is highlighted in the previous paragraph, because it is an adverb that was particularly cherished by Steve Jobs[2]. And it is no coincidence that

it appears on the title of this month's Library choice: "Revolution in the Valley: The Insanely Great Story of How the Mac Was Made,"[3] by Andy Hertzfeld[4], a noted member of the original Apple Macintosh team.

This book was published by O'Reilly in 2005, precisely 21 years after the original Macintosh was released, marking the actual "Twentieth Anniversary" celebration.

Mr. Hertzfeld first poured the memories onto a web page in early 2004, still available at the time of this writing, with the tantalizing and peculiar domain name Folklore.org[5]. The book is simply a beautiful printed copy of the material on that website, and another example of a "coffee table book," just like "The Computer"[6] by Jens Müller and Julius Wiedemann or "Home Computers"[7] by Alex Wiltshire and John Short, two beautiful and interesting books we have talked about in previous editions of this magazine.

"Revolution in the Valley" is as fun to read as it is to explore, telling the story of the making of the Macintosh in short *vignettes*. Each of the chapters, which to a large degree can be read separately from one another, focuses on this or that particular aspect of the final product. The making of the software, the signatures of the team inside the enclosure, the clash between Steve Jobs and Adam Osborne[8], the evolution of the iconography by Susan Kare[9], the disk drive, Microsoft BASIC[10] (or its lack thereof), Burrell Smith[11]'s passion for pineapple pizza, the meaning of the "Real Artists Ship" mantra, and so much more.

The book expands on the website in one particular aspect, making it a gem and a treasure to hold and explore: the photography. The illustrations are sumptuous, from early paper sketches of various ideas to the development of prototypes to magazine covers and even printouts of drawings made with early versions of MacPaint[12].

And yes, there are a lot of coding shenanigans spread throughout the book (we know our audience very well, see?), including how Andy Hertzfeld thought that lines of code were a meaningless metric (and rightfully so), the use of the Hungarian notation developed by Charles Simonyi[13] from Microsoft, which was quite on vogue at the time, or how one Bill Atkinson[14] programming year was equivalent,

according to Steve Jobs, to six standard man-years of coding (and how Hertzfeld thought this was a modest estimate).

The first Mac was, in hindsight, quite a watershed moment for the industry, so advanced for its time, that it took Microsoft 10 years from that point to release a proper user interface on their own operating system. Simultaneously, Apple was busy becoming almost extinct, only to be spectacularly reborn again in the 2000s, but that is a different folklore for somebody else to talk about.

Cover photo by the author.

References

[1] https://en.wikipedia.org/wiki/Twentieth_Anniversary_Macintosh
[2] https://deprogrammaticaipsum.com/steve-gets-trolled/
[3] https://en.wikipedia.org/wiki/Revolution_in_the_Valley
[4] https://en.wikipedia.org/wiki/Andy_Hertzfeld
[5] https://www.folklore.org/
[6] https://deprogrammaticaipsum.com/jens-muller-and-julius-wiedemann/
[7] https://deprogrammaticaipsum.com/alex-wiltshire-john-short/
[8] https://en.wikipedia.org/wiki/Adam_Osborne
[9] https://en.wikipedia.org/wiki/Susan_Kare
[10] https://deprogrammaticaipsum.com/programming-the-liberal-arts/
[11] https://en.wikipedia.org/wiki/Burrell_Smith
[12] https://en.wikipedia.org/wiki/MacPaint
[13] https://en.wikipedia.org/wiki/Charles_Simonyi
[14] https://en.wikipedia.org/wiki/Bill_Atkinson