Issue 014: Minimalism

Graham Lee

November 4th, 2019

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

- Graham argues that we do not need more computing power[1] than that of a 25 dollar computer nowadays.
- Adrian explains why "complex" is better than "complicated."[2]

Enjoy this issue! Please subscribe to our free newsletter[3] to stay updated about new releases.

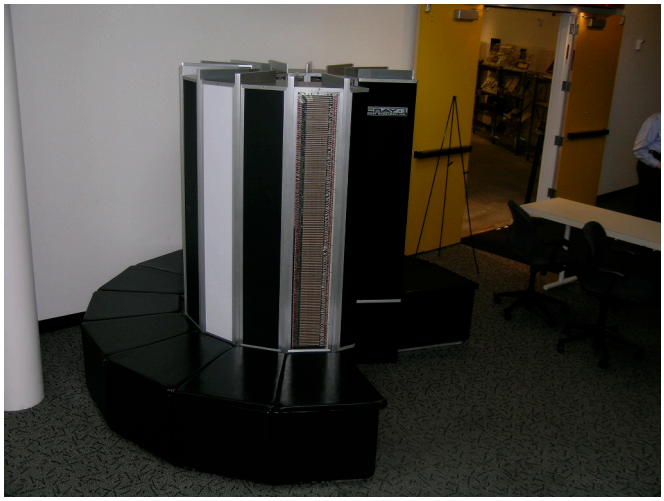
Cover photo by Igor Son[4] on Unsplash[5].

[1] https://deprogrammaticaipsum.com/do-i-need-a-supercomputer/
[2] https://deprogrammaticaipsum.com/complex-vs-complicated/
[3] https://deprogrammaticaipsum.com/newsletter/
[4] https://unsplash.com/@igorson?utm_source=unsplash&utm_medium=referral&utm_content=creditCopyText
[5] https://unsplash.com/s/photos/minimalism?utm_source=unsplash&utm_medium=referral&utm_content=creditCopyText

Do I Need A Supercomputer?

Graham Lee

November 4th, 2019



When Apple introduced their new line of Power Mac G4 computers in 1999, they famously made much of their classification[1] as a supercomputer[2] under the export controls legislation in the United States at the time. Processors (whether CPUs, GPUs, or application-specific circuitry like TPUs or the T2 security processor in modern Macs) have become both faster and more parallelized since then. Many of the chips in a modern smartphone are much more powerful than the Motorola PowerPC 7400 that Apple claimed was weapons-grade.

A feature that Power Mac G4 CPUs, modern smartphone processors, and cores in the supercomputers on the TOP500[3] list have in common is that they spend most of their time waiting. However, they do this waiting for different reasons.

Ken Batcher, an electrical engineer who designed supercomputers for Goodyear Aerospace and discovered parallel sort algorithms, quipped that supercomputers were tools for turning compute-bound problems into I/O-bound problems. In other words, you put so many processors into a supercomputer that the difficulty becomes getting the data between the storage and the processors. The maths is the quick part.

These days, it's fairer to say that supercomputers are tools for turning compute-bound problems into memory latency-bound problems. Not only have processors (of any strip) become faster, but multiple cores, SIMD (Single Instruction, Multiple Data) processing and other techniques have multiplied out the number of instructions that can be running at once in a single silicon package a hundredfold or more. You can easily put a few terabytes of RAM into a single node of a supercomputer cluster, then have to wait for the data you need to get all the way from there through a rat's nest of different caches to the logic unit in the chip. Any failure in the cache control logic—or correct functioning of a caching strategy that doesn't suit your code—increases the amount of waiting.

[1] https://www.techjunkie.com/apples-1999-power-mac-g4-really-classified-weapon/
[2] https://www.techjunkie.com/apples-1999-power-mac-g4-really-classified-weapon/
[3] https://www.top500.org

One thing supercomputers don't tend to wait for is problems to solve. The high amount of capital investment, operational expenditure, and rapid depreciation of high-performance computing assets all mean that owners are keen to extract maximum utilization out of them. Clusters frequently expose resource-management middleware such as SLURM[4] which maintains queues of days or even months of pending jobs.

Meanwhile, waiting to be given a task is much of what a modern personal computer (of any size) does. A phone spends most of its time in its owner's pocket, and while a dedicated baseband processor keeps the connection to the cellular network alive the main CPU and the graphics processor have little to do. Advances in semiconductor technology over the years have improved the potential to switch all or part of these processors off most of the time, to save energy.

Desktop computers spend just as much time doing nothing. The online game type racer[5] tells me that on this keyboard, I can touch type at 71wpm. That corresponds to 355 keypresses per minute, approximately six per second, or approximately 320 million CPU cycles per keypress. Every few million cycles, the CPU has to take a character code from the USB HID driver and turn it into a key event to dispatch to the browser application. Every sixtieth of a second, the GPU has to go to the great effort of working out whether, and how, to draw another character on the screen. Sometimes the characters are spaces. And that's just when I'm typing. If I'm thinking, the computer is taxed by blinking the I-bar every second.

Ah, but modern operating systems do so much more than earlier ones, don't they? Not really. It's true that there are many more features in a modern operating system, but that doesn't mean that they've necessarily expanded to make full use of modern core counts or clock rates. If I open a task monitor on the computer at which I'm editing this post, I see 10% CPU use. There are four real cores (eight virtual cores), so less than one CPU is taxed by "waiting". In fact, most of that reported use is accounted for by the activity monitor itself.

So can we get by with lesser hardware? Experience says yes. I recently replaced my desktop computer with an ARM SoC[6], the sort of thing you're more likely to see in a mid-range Android phone. Even on that, the browser is fully responsive on an HD screen, video plays well, and most applications work without feeling sluggish. When you "do something" like compiling or booting the computer, it's observably slower, but even I need a cup of tea every now and then. Most of the time, around 1% of the CPU's capacity is being taxed.

Granted, even this computer is much more powerful than the Power Mac G4 that first got the Clinton administration worried. The CPU is faster, it has more cores, there is more RAM, and the operating system is stored on a relatively quick eMMC. And that's the interesting part of this situation. A $25 tiny board is way over-specified for much desktop computing use, yet still we're pushed into spending $1000-$3000 on even higher specs.

In theory, the "Megahertz Myth" should have died. Apple spent years telling us that we didn't need more MHz, back when their computers had lower MHz than their competitors'. But now they tell us that each iteration of their hardware is "the fastest yet".

In many situations, we don't need the fastest yet, but can only buy the fastest yet. In many situations, updated software doesn't need the fastest yet, except that it's made by hardware companies who want us to buy the fastest yet.

[4] https://slurm.schedmd.com/overview.html
[5] https://play.typeracer.com
[6] https://twitter.com/iwasleeg/status/1138811353622360064

We should lead the fight against unnecessary obsolescence by example. We should do it because it's a better use of limited environmental resources, and we should do it because it includes more people in the information revolution. Replace your overpowered desktop with a $25 SoC, or even better don't replace it again, ever. Choose software that works on smaller computers, or older computers, or older, smaller computers. Advocate for the adoption of that software on those computers wherever possible.

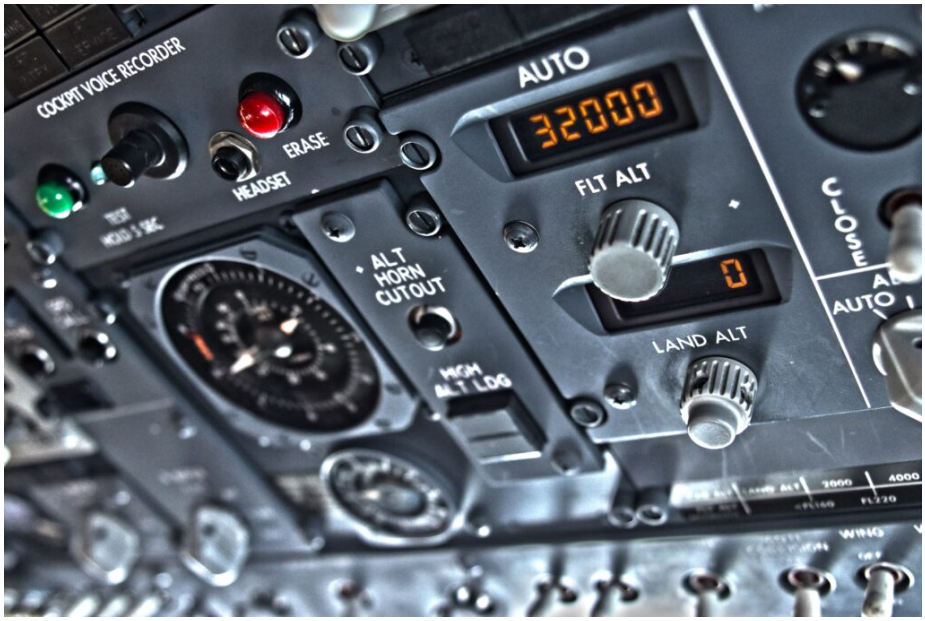
Photo by vonguard on Flickr[7].

[7]https://flickr.com/photos/vonguard/289999533/in/photolist-rCjKX-qzfqJj-eP2Mxw-6cU17S-67nvod-a5i4z-4vi9Gg-2Rrhy-mE5W2-aF4oH5-4vndBU-f9SYrF-io1qcq-4J4dCo-8zGuqU-25L8Zab-5uTur1-cEZfVW-gz1nSz-p3mvTD-QwpEDr-5C1Ekd-9UFSap-KUDgu-482qU2-4vi9EB-bx2JeY-NR1zA-fjHcAE-23K4pUn-wRuBZ-7yfQSN-5Dd3hU-24dPWdf-3GSt2D-bYuoDS-c2on7C-6DaKF-9pFjE-ufUJ8r-aufNM-4vi9Vv-4zyr9h-bKWsjX-bKWsZM-bKWvhi-QPZLe-oQLPEB-68qi1X-a55xg8

Complex Vs. Complicated

Adrian Kosmaczewski

November 4th, 2019



Do you know the difference between "complex" and "complicated"? Most of us certainly mix both concepts, sometimes even several times a day. Spoiler alert: they do not mean what you think they mean.

Being an etymology geek, I opened my beloved "Dictionnaire Historique de la Langue Française[1]," this monumental work by Alain Rey published chez Éditions Robert, and found out the ancestors of both words. Not just because doing it in French makes everything look more intellectual, mind you.

> *Complex* is borrowed from the Latin *complexus*, meaning "made of intertwined elements". This word is the past participle adjectived of *complecti* ("embrace, understand") itself derived from *plectere* ("bend, intertwine"). This word has been used since the XVI century to qualify that which is made of heterogenous elements.

Complex shares the same root (*plectere*) with the words "perplex" (in use since the 14th century in France) and the medical term "plexus" meaning "interlacing" and used since the 16th century as a medical term for "network of nerves or blood vessels".

One of the most common uses of the word in science is the concept of "complex numbers", created by the Italian mathematicians Gerolamo Cardano and Raffaele Bombelli (between 1545 and 1560), later rigourously defined by William Rowan Hamilton in 1828, born from the need to solve the problem of the square root of -1.

On the other hand, "Complicated" has a similar origin but a different construction:

[1] https://fr.wikipedia.org/wiki/Dictionnaire_historique_de_la_langue_fran%C3%A7aise

> *Complicated* comes from the Latin *complicare*, literally meaning "to fold by rolling up". Figuratively speaking this was taken as close to the notion of embarrassment or awkwardness. The word is composed of the word *plicare* which means "to fold".

The Spanish word "plegar" and the French "plier" are direct descendants of this Latin root.

To summarise, "Complex" and "Complicated" stem from slightly different roots: the Latin root *plectere* ("to intertwine") in the former, and *plicare* ("to fold") for the latter.

Complex conveys the idea of a network of intertwined objects, whose state and behavior are continuously modified by their peers in that network. The word *complicated* implies an intrinsic apparent "obscurity" through folding unto itself, inviting to a simple "unfolding" discovery process.

Or, put in another way: complex numbers are not complicated. They are made of real and imaginary parts. Maths can be complicated, though.

In page 12 of the "NeXT Object-Oriented Programming and the Objective-C Language[2]" book, available at the GNUStep project website[3], the image of a pocket watch conveys an explanation to the abstract concept of "encapsulation." Encapsulation of what? Of complication, precisely.

Watches commonly known as "Complications" (such as the Patek Philippe Calibre 89[4], the Franck Muller Aeternitas Mega[5] and the "Référence 57260" de Vacheron Constantin[6]) are complicated, yet not complex, machines. An "Object" in the sense of OOP, should a complicated, but not a complex, thing. Not exactly the experience reported by most code reviewers I have met so far. (By the way, these days the word "complication" is still associated to watches, albeit meaning something entirely different[7].)

The Kubernetes-fueled *microservices* craze has everyone rewriting complicated monoliths into complex cloud-native apps. As my partner in crime Graham once famously wrote in his opus "OOP The Easy Way[8]," microservices is nothing but a new way to say object-orientation. Loosely-coupled entities communicating with each other with messages, anyone?

Simple entities sending messages to each other. Like living cells using mRNA[9] to transfer protein pattern information from the DNA to the ribosome. Uncomplicated, minimalistic entities, used to build larger, more complex systems. Have we forgotten about that? Do we need to learn this over and over?

Software developers have a love / hate relationship with complicatedness, and an almost instant disdain for (if not outright ignorance of) complexity. Complicated systems are great to brag about on Reddit; maintainers also cry about them in private.

A "best practice" has one and only one basic job: to help engineers translate complicated into complex. Most software-related disasters are caused by a simple fact: because of dead-

[2] http://www.gnustep.org/resources/documentation/ObjectivCBook.pdf
[3] http://www.gnustep.org/
[4] https://en.wikipedia.org/wiki/Patek_Philippe_Calibre_89
[5] https://www.franckmuller.com/aeternitas/
[6] http://reference57260.vacheron-constantin.com/fr/la-montre-la-plus-compliquee-au-monde
[7] https://www.imore.com/how-add-third-party-complications-your-apple-watch
[8] https://leanpub.com/ooptheeasyway/
[9] https://en.wikipedia.org/wiki/Messenger_RNA

lines, managers, tooling, or just plain ignorance, software developers have a tendency to build complicated, instead of complex, systems.

Let us write and run complex, not complicated, systems. We cannot get away from complexity; that is our job as engineers. But we can drop the complicated bit, and minimalism will follow.

Instead of running a 500 MB Docker container with Linux, Apache, PHP, and the full source code of that monolith written in 2002, let us instead run a 5 MB container built `FROM scratch` running a microservice written with Go[10].

Instead of running Windows, macOS or Ubuntu, let us boot our PCs with KolibriOS[11], Tiny Core Linux[12] or Damn Small Linux[13]. Or even with Minix[14], MikeOS[15], or FreeDOS[16].

Instead of writing mobile apps with yet another hyped layer of abstraction, use the lower-level language recommended by the platform vendor, giving you direct access to the native APIs of the device, and a tighter control of the memory and CPU consumption of your app.

Instead of using yet another programming language several dozen abstraction layers away from the actual platform, let us go back down a bit, thanks to languages like Go[17], Rust[18], D[19], Crystal[20], or Elixir[21]. Thanks to projects like LLVM[22], many high-level programming languages do not need need a runtime layer anymore, can be cross-compiled, and offer blazingly fast performance.

Instead of Emacs, let us try Joe's Own Editor[23].

And after having launched it, let us write our own operating system[24] with it.

Cover photo by Dan Lohmar[25] on Unsplash[26].

[10] https://golang.org/
[11] http://kolibrios.org/
[12] http://tinycorelinux.net/
[13] http://www.damnsmalllinux.org/
[14] http://www.minix3.org/
[15] http://mikeos.sourceforge.net/
[16] https://www.freedos.org/
[17] https://golang.org/
[18] https://www.rust-lang.org/
[19] https://dlang.org/index.html
[20] https://crystal-lang.org/
[21] https://elixir-lang.org/
[22] http://llvm.org/
[23] https://en.wikipedia.org/wiki/Joe's_Own_Editor
[24] http://mikeos.sourceforge.net/write-your-own-os.html
[25] https://unsplash.com/@dlohmar?utm_source=unsplash&utm_medium=referral&utm_content=creditCopyText
[26] https://unsplash.com/s/photos/complex?utm_source=unsplash&utm_medium=referral&utm_content=creditCopyText