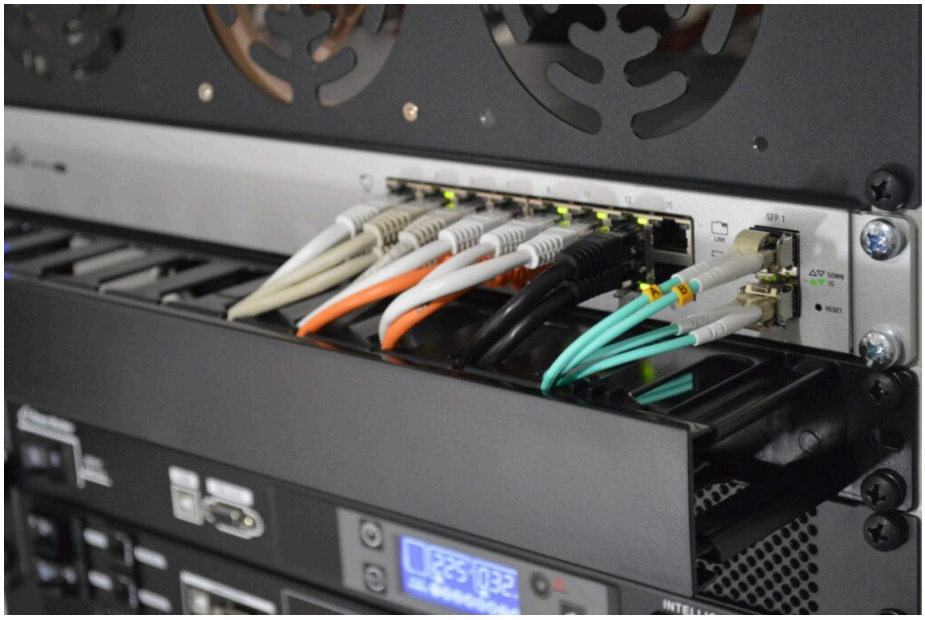
Issue 027: Networking

Adrian Kosmaczewski

December 7th, 2020



Welcome to the twenty-seventh issue of *De Programmatica Ipsum*, dedicated to the subject of *Networking*. In this edition:

- Graham explores one of the most mysterious slogans of all time[1] in the computer industry.
- Adrian recalls his first steps on the Internet[2] with a capital letter.
- In the Library section[3], Graham peruses two of the most important works of Steve McConnell[4], namely *"Code Complete, 2nd Edition"* and *"More Effective Agile"*.

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

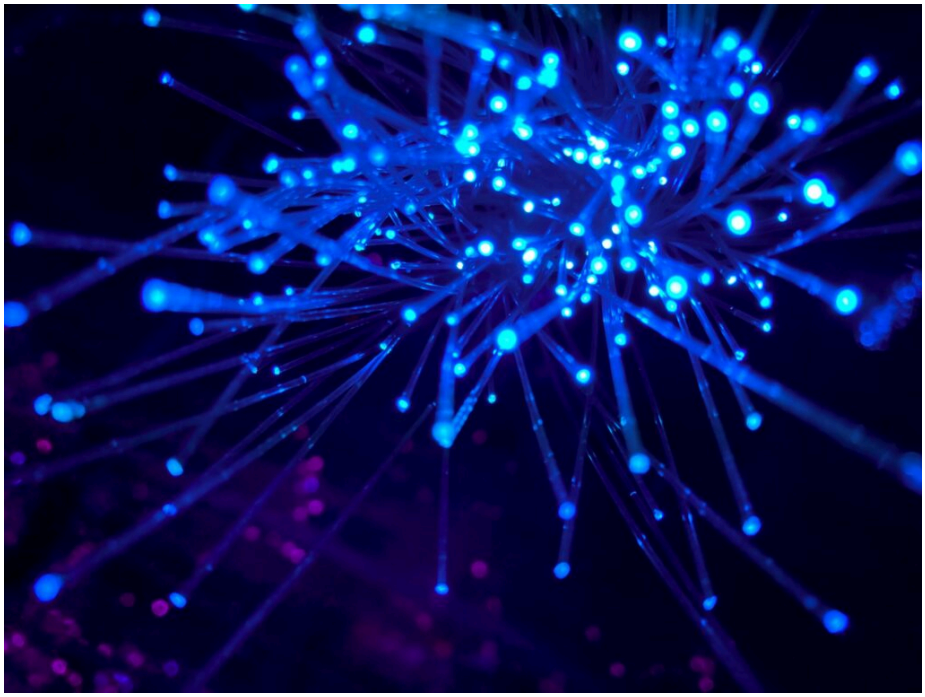
Cover photo by Thomas Jensen[7] on Unsplash[8].

[1] https://deprogrammaticaipsum.com/the-network-is-the-computer/
[2] https://deprogrammaticaipsum.com/sniffing-packets/
[3] https://deprogrammaticaipsum.com/category/library/
[4] https://deprogrammaticaipsum.com/steve-mcconnell/
[5] https://deprogrammaticaipsum.com/newsletter/
[6] https://deprogrammaticaipsum.com/contribute/
[7] https://unsplash.com/@thomasjsn?utm_source=unsplash&utm_medium=referral&utm_content=creditCopyText
[8] https://unsplash.com/s/photos/network-cables?utm_source=unsplash&utm_medium=referral&utm_content=creditCopyText

The Network Is The Computer

Graham Lee

December 7th, 2020



Back in the mists of time, an early Sun Microsystems employee by the name of John Gage coined the term "the network is the computer" to describe the centrality Sun put on network capabilities when designing their workstations.

As we saw last month in the hardware issue, Sun and other companies were interested in taking existing time-sharing computer designs, and selling customers one such system each. Given that the design of the computer means that multiple people can share one system, you need some way to justify this over-provision of resources. Move attention to the network, then: if everybody can login to access their files, printers, and emails from any of the workstations you deploy, then IT administration becomes part of the facilities department rather than the, well, IT department. The culmination of this design was the SunRay "ultra-thin" client, though that meant Sun had gone back to selling you a single time-sharing computer and multiple access terminals.

Note that Sun were careful to say *the* network, not *our* network. Silicon Valley has an egregious history of taking publicly-funded work (i.e. work paid for by US taxes) and enclosing it to create multiple billions of dollars in profit for a few individuals. The most obvious modern example is Google, who commercialised work that was in part funded by NASA and the NSF (as credited in the paper "The Anatomy of a Large-Scale Hypertextual Web Search Engine").

Sun were no different. In a display of hubris, the founders named the company after the Stanford University Networks project, an effort to connect computers across the campus to

the ARPAnet via TCP/IP over Ethernet. Andy Bechtolsheim's work in designing the SUN workstation was paid for by DARPA and the university's CS department.

Sun were not the only people to take publicly-funded work from the Stanford campus network and exploit it for private gain. Sandy Lerner, the director of computing facilities in Stanford's business school, would be forced to resign her position at the university for commercially selling a university-designed network appliance that she had no license to exploit. She subsequently founded a private company to do the same: the Stanford "blue box" became IOS and the company's name was Cisco.

Anyway, Sun shortened the rather dull phrase "somebody else's network is the computer we want to profit from" and a marketing catchphrase was born. Oracle (finally a company bootstrapped on the work of its founder) evidently did not agree, or at least could not use the phrase to sue Google, and did not renew the trademark registration on acquiring Sun. Enter Cloudflare, who swooped in with much fanfare to snipe the phrase and use it for something something internet of things.

There is actually a lot packed into that slogan, "the network is the computer", and lots of information for the software designer. To understand why, let us explore the truth of the statement: *why* is the network the computer?

It has all to do with Conway's Law. As frequently stated, Conway's Law[1] says that organisations design systems that mirror their communication structure. This is usually interpreted very narrowly, meaning that software is designed along the communication structure of the software development team. In fact, Melvin Conway expressed that the word "systems" should be "defined broadly".

It is common to think of a software product as being a technical system that mediates between two social systems: the system of producers (i.e. the software team) and the system of consumers (i.e. the clients, customers, "users"). In fact, this view comes from the Taylorist image of software creation that was prevalent in the twentieth century. The client's roles are to know and express what software they need, and to accept our software once we have implemented all their requirements. Our role is to ask the questions that will uncover the requirements correctly and efficiently, and then to mechanistically and efficiently convert those requirements into the expected software.

A holistic and more accurate view of software development sees all three of these "systems" as part of a single social-technical system, in which the interactions between producers and consumers are too important, too frequent, and too nuanced to be externalised as outside factors influencing distinct systems. The software (both its existence and the act of its production) mediates a dialogue between developers and users over the software's behaviour, in addition to producing a dialectic in which the appearance of the software both challenges existing beliefs about the system in which it is embedded, and shapes a new reality.

An early attempt to move away from the "three systems" model toward the "one system" model can be seen in the Manifesto for Agile Software Development[2]. The manifesto maintains two distinct "classes" of actor—the "developer" who has the skill to create software and sells their labour into the software-creation exercise, and the "business" defined as sponsors who control the means by which software is paid for (gold owners) and users who supply the reason for the software's existence (goal donors). The developers are still the knowledge-work equivalent of hired muscle:

[1] https://en.wikipedia.org/wiki/Conway's_law
[2] https://agilemanifesto.org

- Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.
- Agile processes harness change for the customer's competitive advantage.
- Working software is the primary measure of progress.

Nonetheless, the Agile manifesto describes the act of software creation as an ongoing dialogue between these actors, and requests a fuzzier and more fluid boundary between their domains. Thus, it includes principles describing their interaction:

- Business people and developers must work together daily throughout the project.
- Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.

Notice the tension inherent in the manifesto's principles, between the boundless (more working software, faster, is better) and the bounded (we should be able to reproduce this working situation forever). Developers working in the Agile context are invited to be Animal Farm's Boxer, working as hard as they can for as long as they can in the hope to avoid the glue factory.

A more progressive paradigm for software development would remove the classifications of "programmer" and "business people", regarding all as individuals with particular expertise who come together to pool that expertise in pursuit of a common goal. And thus could the network as the computer be truly recognised: the network as distributed source of and access to computing and communication supporting the social system in which it is embedded to the greatest possible extent. Let us explore what that would look like.

The Internet Protocol was designed in the United States during the 1970s, at the military Advanced Research Projects Agency (ARPA). As the designers were American cyberneticians, the system contains design principles that would be desirable in an ideal system in contemporary America. Each node is a "free actor", choosing to form peer relationships with other nodes and route packets based on local information about those nodes and the destinations of the packets. The cyber-utopia, evinced by fanzines, the Whole Earth Catalog, and numerous cyberpunk novels, is one of total anarchy and individual liberty. The internet—as designed, anyway—perceives censorship as damage and routes around it.

In fact the period of establishment of IP as the de facto standard for internetworking (in the UK's Joint Academic Network, Janet, the "killer app" that saw IP winning out was the X remote desktop protocol) also saw the rise and entrenchment of global capitalism and the proliferation of projects designed to transfer wealth en masse to the small cohort of billionaires. Ideas of interactions in "the marketplace" changed, and thus so did the ideas about a network best configured to support them. Out was the Kevin Flynn who ran an independent arcade business and deserved restitution from Encom in the movie Tron. In was the Encom that was still dominant in Tron: Legacy and released Encom OS 12 with the killer feature: "this year, we put a 12 on the box".

As the ideology behind the "free market" changed, so did the "marketplace of connectivity": the internet perceived censorship as an opportunity and asked who was to pay for it. Thus the modern internet: a small number of global players controlling the core, and a ragtag assortment of startups wondering whether to give their seed money to Microsoft, Google, or Amazon in exchange for a seat at the table. The network is the computer, and the computer has five access terminals[3].

[3] https://deprogrammaticaipsum.com/five-computers/

To understand how the internet's adaptability to prevailing ideas about social systems enabled its survival, turn to a similar system that did not achieve the same. The Soviet vision of the internet, designed by cyberneticians in research institutions around the Union, never became a reality. Its story is told in the book How Not to Network a Nation[4] and the article InterNyet: why the Soviet Union did not build a nationwide computer network[5]. Where Vint Cerf and Bob Kahn were supported in their plans to introduce a nationwide military computer network, their counterpart Viktor Glushkov (a Russian-born mathematician who founded the Institute of Cybernetics in Kyiv, Ukraine) met political obstacles and funding cuts. Why?

Principally because the network was *not* the computer: the system proposed did not support the system that was in place. Where the ARPA vision of the internet promised a decentralised system (what Americans think their system is) and was delivered by the heavily centralised, bureaucratic, planned economy of the US military apparatus (what America really is), the Glushkov project OGAS promised exactly the same in a very different context. The OGAS internet would be a decentralised economic planning system (when Soviet political leaders believed that they had, or should have, total control over a centralised economy), delivered by a heavily centralised, bureaucratic, planned economy (which did not really exist: spending was much more regional and decentralised in the Soviet Union, which relied on local knowledge and mutual assistance between lower-down functionaries who gave the *appearance* of implementing a central plan).

So for a software system to succeed, the network must be the computer: the physical availability and distribution of the platform that provides access to the software (i.e. the computer) must reflect, support, and reproduce the interpersonal connections and data exchanges that will achieve the goals of the people involved (the network). The computer can change the network, and the network can change the computer, but this will happen with mutual involvement. Software designers must expand their understanding of Conway's Law.

Photo by JJ Ying[6] on Unsplash[7]

[4] https://mitpress.mit.edu/books/how-not-network-nation

[5] https://web.mit.edu/slava/homepage/articles/Gerovitch-InterNyet.pdf

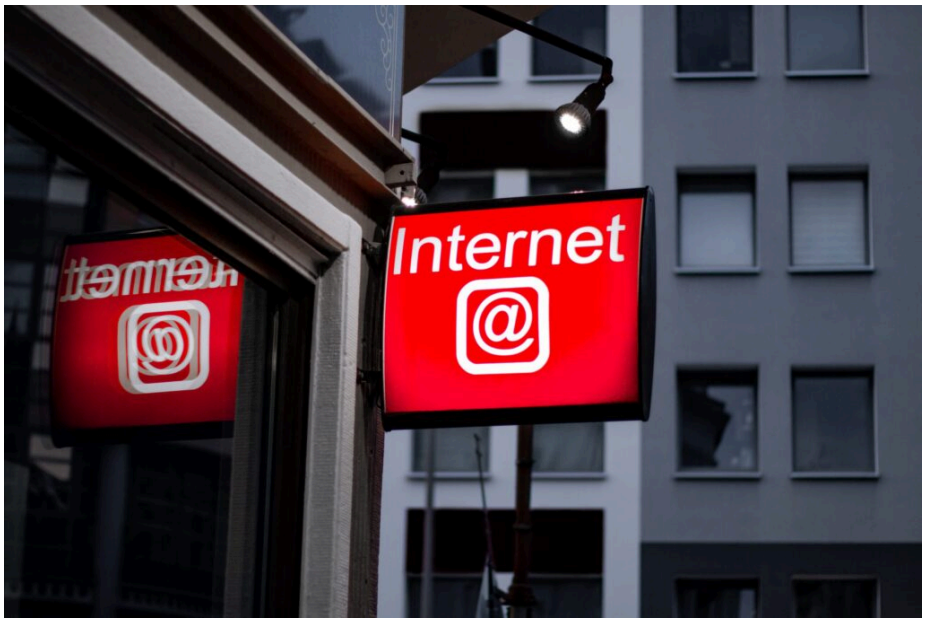
[6] https://unsplash.com/@jjying?utm_source=unsplash&utm_medium=referral&utm_content=creditCopyText

[7] https://unsplash.com/s/photos/network?utm_source=unsplash&utm_medium=referral&utm_content=creditCopyText

Sniffing Packets

Adrian Kosmaczewski

December 7th, 2020



I remember vividly the first time I saw somebody "online." It was early in a morning of December 1994, in the hallways of the "Sciences 2" building of the University of Geneva[1]. One of my classmates, who worked part-time as a professional software developer, was connected to a terminal with glowing yellow text over a dark brown background. To my question about what he was doing, he answered with a simple "I am downloading stuff from Apple's server in California."

There were so many things in that short phrase that I just could not immediately understand.

To begin with, international phone communications were insanely expensive in 1994. To the point that I regularly exchanged written letters (you know, the paper and ink kind) with the friends I had left behind in Argentina when I moved to Switzerland in 1991. A simple letter to Buenos Aires cost me 1.50 Swiss Francs back then; a direct phone call could easily rack tens of Swiss Francs per minute, far beyond my budget. Understandably enough, my first reaction was to think that my friend's endeavours were going to cost him a fortune.

Fear not, he explained, "for I am connected over The Internet."

Say what?

That short interaction was the first time I ever heard the name of the allmighty Network of Networks, the clumsily-named Information Superhighway, the future-proof Cyberspace.

I learnt, shortly after, that most universities were connected to one another with high-speed links, and that they exchanged research, data, documents, and even people sent each other "e-mail" (whatever that was) for a fraction of the cost of common telecommunication channels.

[1] https://www.unige.ch/sciences/en/

As a student I could be a part of it. Thus, a few days later I got my own student account: my first Internet access ever, with the username `kosmacze`. It required me to login through small terminals, connected to a VAX[2] minicomputer somewhere in the campus, itself running a version of VMS[3].

It is through that gloomy terminal that I discovered the joys of downloading shareware from ftp.funet.fi[4]. It is there that I discovered mail[5], Usenet[6], Gopher[7] (no, not the Go language mascot), talk[8] (the ancestor to Microsoft Messenger[9] and WhatsApp), and shortly after, a clickable thing called the World Wide Web, apparently created not far from where I was studying[10], in a NeXT workstation somewhere in the depths of CERN. Not far at all; just a 45 minute ride on bus 15 (there were no tramways back then on that line,) and boom, you have arrived to CERN.

Not long after that, I had lost almost all interest in my studies (I was supposed to graduate in Physics) and the 'net became the greatest interest (and most important outcome) of my academic life.

It became an addiction, to be honest.

Around Autumn 1995 I saw a demo of Netscape Navigator[11] in action; and thus I decided that I wanted to hook my home PC to the Internet. I downloaded Netscape for Windows 3.1 in the university campus, fled home, inserted the floppy disk, launched the app, and I was greeted with a laconic dialog box telling me that I did not have a thing[12] called `winsock.dll`.

Say what?

It turns out that my operating system (well, if you dare call Windows 3.1[13] such a thing) did not include any kind of networking support. Of course I did not know that, and even better, I had no idea how to add it, and thus how to get my PC connected "online". After a few questions on Usenet (which in many ways worked as a precursor to Stack Overflow) I ended up downloading Trumpet Winsock[14], and lo and behold, Netscape this time loaded without problems.

But I did not have a modem, and quickly found out that Netscape was not yet very useful without one.

To make a long story short, at some point during early 1996 I bought a cheap 14.4 kilobit per second modem, taking 10 minutes in average to download a single megabyte. Together with a monthly subscription to the new "Blue Window" service by Telecom PTT[15] (soon to be renamed Swisscom), lo and behold, I was online.

[2] https://en.wikipedia.org/wiki/VAX
[3] https://en.wikipedia.org/wiki/OpenVMS
[4] http://ftp.funet.fi/
[5] https://kb.iu.edu/d/agax
[6] https://en.wikipedia.org/wiki/Usenet
[7] https://en.wikipedia.org/wiki/Gopher_(protocol)
[8] https://en.wikipedia.org/wiki/Talk_(software)
[9] https://en.wikipedia.org/wiki/Microsoft_Messenger_service
[10] https://home.cern/science/computing/birth-web
[11] https://en.wikipedia.org/wiki/Netscape_Navigator
[12] https://kb.iu.edu/d/advk
[13] https://en.wikipedia.org/wiki/Windows_3.1x
[14] https://winworldpc.com/product/trumpet-winsock/3x
[15] https://www.swisscom.ch/en/about/company/portrait/history.html

Amazon[16] was already there; instead of Google, one had Yahoo![17] and AltaVista[18]; instead of Tumblr, there was GeoCities[19]; no Gmail but HoTMaiL[20] (yes, that was the original spelling.) Hotmail was quickly swallowed by Microsoft, became Passport, and now it is part of Office Online. Apparently though, using your @hotmail.com e-mail can be troublesome[21] these days.

On August 29th, 1996, around 16:00 CET, I published my first web page. I remember precisely the moment. The mandatory "visit counter" stayed stubbornly low until I submitted my website to the AltaVista crawler. That early website, one of the 250'000 websites[22] available on the web at that time, featured prominent characteristics from that era: horrendous background colors, animated GIFs, and rather simple font choices (Times New Roman, thank you so much) but, who cared. I was online.

I learnt HTML reading Elizabeth Castro's magnificent "Visual Quickstart Guide"[23]; I had not yet learnt any JavaScript, and CSS did not exist yet. Just use `<TABLE>` for layout and `<FONT>` tags to style your content, good luck.

As I said, I was online. I edited my files using HoTMetaL Pro[24], uploaded them with WS_FTP[25], all while listening to music playing in Mod4Win[26], or some Internet radio playing through RealAudio[27].

By 1997 I was already working as a "Web Developer" (that is what people called "Fullstack Developers" back then). Those were the times of EditPlus[28], ICQ[29], Photoshop 4[30], the ground-breaking Macromedia Dreamweaver[31], and of course its clumsy competitor, Microsoft FrontPage[32], also known in the industry as a generator of random HTML tag soups.

My employer allowed me to work from home (yes, in 1997) and for that matter provided me with a 56 kilobits per second 3Com[33] modem, capable of downloading a whooping megabyte in just under 3 minutes. They even paid for a separate phone line just for being connected on the Internet at all times.

Looked from the perspective of the OSI model[34], my career started on the highest level of the hierarchy; the Application Layer[35], distributing content through its most prominent

[16] https://www.webdesignmuseum.org/gallery/amazon-1995
[17] https://www.webdesignmuseum.org/timeline/yahoo-1996
[18] https://www.webdesignmuseum.org/timeline/altavista-1996
[19] https://www.webdesignmuseum.org/timeline/geocities-1996
[20] https://www.webdesignmuseum.org/timeline/hotmail-1998
[21] https://www.poynter.org/tech-tools/2018/not-so-hotmail-what-your-vintage-email-address-says-to-potential-employers-2/
[22] https://www.internetlivestats.com/total-number-of-websites/
[23] https://www.amazon.de/gp/product/0201884488/ref=dbs_a_def_rwt_bibl_vppi_i35
[24] https://en.wikipedia.org/wiki/HoTMetaL
[25] https://en.wikipedia.org/wiki/WS_FTP
[26] http://kay-bruns.de/mod4win/
[27] https://en.wikipedia.org/wiki/RealAudio
[28] https://www.editplus.com/
[29] https://icq.com/
[30] https://www.webdesignmuseum.org/old-software/graphic-software/adobe-photoshop-4-0
[31] https://www.webdesignmuseum.org/old-software/html-editors/macromedia-dreamweaver-1-2
[32] https://www.webdesignmuseum.org/old-software/html-editors/microsoft-frontpage-97
[33] https://en.wikipedia.org/wiki/3Com
[34] https://en.wikipedia.org/wiki/OSI_model
[35] https://en.wikipedia.org/wiki/Application_layer

celebrity, the HTTP protocol[36]. So now I was ready to dive deeper.

Exactly two years ago I told the story[37] of how I took my first steps into packet sniffing, circa 2000:

> I found a copy of CaptureNet, a freeware packet sniffer part of the SpyNet/PeepNet by Laurentiu Nicula[38]; then I looked up for the port number used by MSN Messenger[39] (it was 1863 in case you were wondering.) Finally I found out how to enable "promiscuous mode" in the network card in my laptop. (...) Instantaneously, my screen started to show me the conversations my peers were having on MSN Messenger. And I mean all of it. (...) All on my screen, ready to read, without any encryption. After changing the sniffing port to 80, I used CaptureNet's uncanny feature of reconstructing the web pages. All of my colleagues browsing at that very moment (...) appeared in my laptop.

The relation between networking and security is obvious and painful. OpenBSD was designed to be secure from the ground up: back in 2001[40], that meant all "all non-essential services are disabled" by default, including network services. Unlike, say, Windows and its port 139 (aka NetBIOS) ready to welcome visits from Back Orifice[41] users.

In 2002 I bought an iBook, and I downloaded Apple's developer tools over my Swisscom-provided ADSL line. It took me 3 hours to get those 300 MB of IDEs, documentation (when it was still excellent[42]), tools, and libraries; that is about 36 seconds per megabyte. That was 15 times faster than in 1996.

In 2020 I can `npm install` JavaScript packages at a sustained rate of 63 MB per second; that is, around 2300 times faster than in 2002, or around 35'000 times faster than 24 years ago. Always over yet another, this time more powerful, Swisscom-provided connection.

These days we can run Windows 2000[43] in our browsers, all virtual. In the case of Docker containers and Kubernetes clusters, networks are virtual, just like the machines connected through them. An infinite regression of (hopefully) encrypted packets traveling from one location in memory to another. We curl[44] and ping and gping[45] and scp, even if we should not[46]. It is virtual until it is not.

These days huge proportions of mankind are "downloading stuff from Apple's server in California" continuously, all the time. Macs are even uploading stuff[47] continuously, all the time, for whatever reason.

In the world of the pandemic, our TVs are directly (or indirectly) connected to the Internet. Our cellphones have become Internet terminals. This magazine is on the Internet, and some of you contribute to it[48] sending money over the Internet. We buy our groceries and books

[36] https://en.wikipedia.org/wiki/Hypertext_Transfer_Protocol
[37] https://deprogrammaticaipsum.com/the-weakest-link/
[38] https://www.linkedin.com/in/mendark/
[39] https://en.wikipedia.org/wiki/Windows_Live_Messenger
[40] https://www.zdnet.com/article/openbsd-the-most-secure-os-around/
[41] https://en.wikipedia.org/wiki/Back_Orifice
[42] https://www.caseyliss.com/2020/11/10/on-apples-pisspoor-documentation
[43] https://bellard.org/jslinux/vm.html?url=win2k.cfg&mem=192&graphic=1&w=1024&h=768
[44] https://curl.se/
[45] https://github.com/orf/gping
[46] https://lwn.net/SubscriberLink/835962/ae41b27bc20699ad/
[47] https://sneak.berlin/20201112/your-computer-isnt-yours/
[48] https://deprogrammaticaipsum.com/contribute/

over the Internet. We have video conferences with our loved ones over the Internet. Many of our jobs literally consist of being on the Internet. Politics happen on the Internet with politicians that do not understand the Internet. Economics and sport news are distributed on the Internet. We watch probes on Mars and astronauts on the ISS live on the Internet. People are harassed and phished on the Internet. Governments struggle and then clumsily decide to regulate the wrong parts of the Internet.

The future is here, and it looks precisely like what many science fiction authors thought it would be. It is a state of addiction, and in a quite ironic twist, we are slowly becoming more and more disconnected from one another[49], sniffing IP[50] packets as if it were a drug.

Cover photo by Leon Seibert[51] on Unsplash[52].

[49] https://www.nytimes.com/2019/03/23/sunday-review/human-contact-luxury-screens.html

[50] https://en.wikipedia.org/wiki/Internet_Protocol

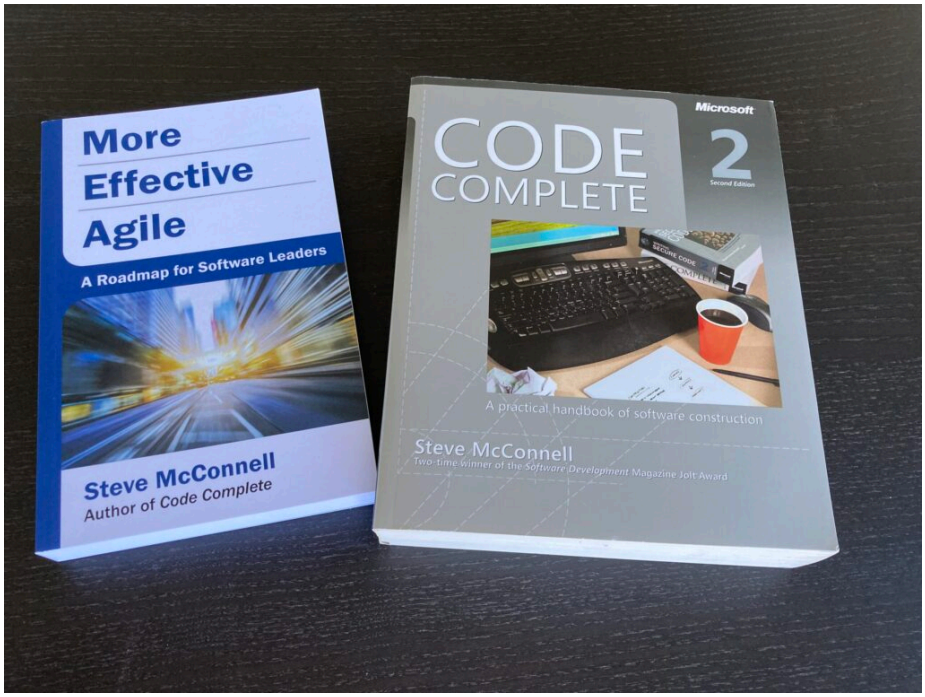
[51] https://unsplash.com/@yapics?utm_source=unsplash&utm_medium=referral&utm_content=creditCopyText

[52] https://unsplash.com/s/photos/internet?utm_source=unsplash&utm_medium=referral&utm_content=creditCopyText

Steve McConnell

Graham Lee

December 7th, 2020



I almost wrote this article not about McConnell, but Microsoft Press. Why? Because developers always have something to learn, books have been a great way to share information for centuries, so reading about computing is central to the software engineering experience. If you do not believe me, reflect on the activity you are undertaking *right now*, reading an online magazine about computing.

You can tell the seriousness with which a platform company treats its developers—or its "developers, developers, developers"—by the tools, knowledge and support it gives them. Where some companies release their API documentation in bound form through a recognised publisher, Microsoft just went and started their own book imprint. Not to bind their header comments between two boards, but to support professional programmers whatever their stripe (though maybe bear in mind which company has your back when choosing your technology, please).

Am I over-selling the importance of Microsoft Press, though? Steve McConnell, in *Code Complete, 2nd Edition* (2004, Microsoft Press), cites DeMarco and Lister's *Peopleware* when he tells readers "one book is more than most programmers read each year." He recommends trying to get through one every two months, as well as journals like *IEEE Software*, *Communications of the ACM*, and the now defunct and much-missed *Dr. Dobb's Journal*. The "software developer's reading plan" at the back of the book contains 5 introductory-level books, 8 intermediate-level, and 7 leadership-level. It would take the median programmer (at least in 1999 when *Peopleware* was written) over 20 years to follow the plan, but a little over three years at McConnell's suggested rate. As long as you read *CC2E* first to know how

quickly you are supposed to be reading!

Coincidentally, I left university to look for a job (which I found in the very room where I conducted my job search) in 2004, the year *CC2E* was published. Within a couple of years, when it was clear that my ability to solve problems using a computer was being hampered by my home-grown and BASIC-honed techniques at expressing computer programs, a manager gave me a copy of *Code Complete*. And the rest, as they say, is history.

There is lots to critique about the book, of course, particularly in 2020. Writing at the start of the Agile revolution, McConnell does not have much to say about agile practices or methodology (two entries in the index, in fact, both pointing to single-paragraph descriptions of books; there is no discussion of agile in the main text). Sure, all of the parts are there—there is a distinct shift in the discussion of change management and risk between the first edition and the second, for example—McConnell acknowledges that the different activities in software construction occur in parallel, and that the earlier a change is handled, the cheaper it will be. But he never quite gets to the conclusion that software need not be "measure twice, cut once"; that unlike carpentry, software has a limitless supply of infinitely malleable wood.

In some of the places where McConnell uses multiple citations to make a claim about software engineering, other writers have found the citations based on questionable data or not generalisable to the way most software is written, particularly "classic" studies from the 1970s looking at programming batch systems. But we will save that story, because a future instalment of the Programmer's Library will cover one of those books in detail. It is more interesting than it sounds!

There is also lots to recommend the book: in fact, in the 16 years since it was written, nothing else comes close. I still use and recommend practices described in *Code Complete*. I have internalised most of the ideas about code organisation and conception. When I see someone who reminds me of mid-aughts me, this is one of the books that springs to mind.

Other books have come and gone that try to do bits of what McConnell does here, with varying degrees of success. Robert Martin's *Clean Code* on low-level design teaches similar ideas but with less intellectual rigour. The same goes for his *Clean Coder* on the practice of being a developer, though this is a more contested field with good showings from Pete McBreen (*Software Craftsmanship: The New Imperative*) and Hunt & Thomas (*The Pragmatic Programmer*). But none of these come close to supplanting *CC2E* as a manual for turning a programmer into a software engineer, and the realities of the software publishing business in 2020 mean that no replacement is likely to show up soon.

If all Steve McConnell had done was write this one book, his would be a significant contribution to the field. Even among the MS press books on software requirements, object thinking (not OOP), secure software development, solutions architecture, and more, this is a standout work. But he was and remains a prolific author, so now we turn to the rest of his library.

We will start by going back from 2004 to 1996. After the first edition of *Code Complete*, McConnell wrote a book about improving the capability of software teams to successfully deliver working software, ways of so doing he had uncovered by doing it and by helping others to do it. This book was *Rapid Development*, as distinct from the then-popular fad of Rapid Application Development. *Rapid Development* was explicitly *not* "programming, motherfucker": an exhortation to just get down to coding as quickly as possible. *Rapid Development* was the (then, and from what I have seen, now too) groundbreaking advice that you should maybe align your development practices with your customers' expectations and needs, and that doing so will help you give them what they need faster. This book had many

recommendations for developers and managers to adopt on their team, including early and continuous delivery of working software through iterative, evolutionary development; continuous collaboration between the software team and the customers throughout the project; frequent feedback on the status and risks and reprioritisation so that the biggest risks are always next to be addressed; and more.

If this all sounds like it actually *is* agile software development, expressed four years before the Snowbird ski trip, that is for multiple reasons. Firstly by the time the manifesto was written, there was already broad understanding (at least among the more experienced and expert practitioners of software delivery) that the ideas of iterative, incremental development were transformative to the capabilities of software teams, there just was not yet a common banner under which all of the people calling for this could march. Secondly it is because I have been selective in my presentation of what is in *Rapid Development*: one of the "case studies" features a team who fail because they integrate their work too early, before all of the modules are fully tested and debugged, and this causes schedule slippage. Nowadays we recognise integration as one of the risks that we *should* mitigate by early and continuous monitoring; thus continuous integration.

But mostly it is because agile software development has been hollowed out since its introduction at the beginning of the millennium. Ask many development teams what they understand by Agile in 2020 and you will get a list similar to the highlights of *Rapid Development* I gave above: incremental and iterative delivery, customer involvement on the team, frequent retrospectives and feedback. That is not the entirety of the story, that is the Project Managers' Declaration of Interdependence[1] (originally published on Alistair Cockburn's site in 2005). Agile software development also included a number of *technical* practices, not explicitly called out in the manifesto, on the basis that what we do changes more frequently than why we do it.

We have already mentioned Continuous Integration, and developers might readily mention Test-Driven Development (and maybe its more agile siblings, Behaviour-Driven Development and Acceptance Test-Driven Development, which remind us that software is done not when the tests pass, but when the software does what the customer wants from it). Some may begrudgingly allow daily stand-up meetings as an agile practice, though they may grumble. But how about user stories? Those are a technical practice, and an example of specification-by-customer-conversation. Domain-Driven Design encourages a shared vocabulary between the software team and the customers, enabling those conversations. A review of agile practices in scientific software development[2] lists 35 practices explicitly mentioned in the Scrum and XP methodologies. Of those, I informally count 22 technical (or at least not purely project-management) practices, and I would also suggest that those technical practices are the ones most patchily adopted across the industry, based on my own experience in many teams that describe their work as "Agile".

Rapid Development gets a bye on some of those practices on the basis that they had not been invented in 1996, but also because McConnell makes a much more general and much more useful argument: technical practices *are not guaranteed* to improve your ability to deliver working software to your customers. Indeed adopting new practices on a new project is likely to *reduce* your control over and understanding of the project, thus *harming* your ability to communicate effectively with customers what you are capable of, how much it will cost, and how long it will take. This incredibly simple and important notion has been rediscovered

[1] https://www.adventureswithagile.com/2014/08/19/declaration-of-interdependence/
[2] https://dl.acm.org/doi/10.1145/1985782.1985784

many times in the industry, and is currently usually cited in the form of "innovation tokens" from Dan McKinley's choose boring technology[3].

CC2E gets a bye on its short shrift on Agile for the innovation tokens reason: before the agile-industrial complex took over and made agile all about hiring scrum masters and producing burn-down charts, it was a shorthand for a lot of project management *and* technical practices and there was not yet compelling evidence that taken together, those practices improved the life or capability of a software engineer.

Maybe there is not, but we can turn back to Steve McConnell for a modern discussion with his 2019 book, More Effective Agile[4]. *More Effective Agile* is the *Rapid Development* 2.0 for the agile-industrial complex, though we should note that the world is in a much better place when it comes to software delivery than it was in 1996. Now the question most people have is not "why do my software projects always fail", but "why do my software projects always cost a bit more, or take a bit longer, than I would like". Eventually many people end up with the question "why has all my forward progress ground to a halt in a mire of technical debt and defect fixing", but often answers to those questions can be deferred for the successor CTO to answer.

In this context, McConnell revisits the central premise of *Rapid Development*: what does a software team need to succeed, are they getting it, and how can they get it? He describes Scrum as a good baseline framework for organisations transitioning to the agile approach, and "scrumbut" as a common reason for novice adopters to fail. Scrumbut is a hybrid methodology, where you treat Scrum as an a la carte selection of practices to pick and choose as you see fit. "We do scrum, but we have our daily standups once a fortnight". "We do scrum, but our scrum master is also our engineering manager".

Does this make McConnell a member of the brinksmanship school of the agile-industrial complex: if agile is not working for you, it is because you are not agiling hard enough? No, it makes him a realist: scrum is a framework for process improvement, but you can only succeed at improving your process once you have measured what is wrong with it and hypothesised how to fix it. If you are just starting out, stick in the shallow end of the pool and adopt the baseline process. One that is working for you and you are not going to drown, it is time to try changing things.

Just as agile itself jettisoned many of the agile practices from methodologies like XP, and *Rapid Development* borrowed from Fred Brooks the idea that there is "no silver bullet", so *More Effective Agile* does not mandate specific technical practices. The two that are given decent space in the book are Continuous Integration and Continuous Delivery, both of which are still unevenly distributed throughout the industry so are good choices for an author who wants to make agile teams more effective.

Steve McConnell has been working in software and teaching software practitioners for over three decades, and there is still much to learn.

Cover photo by Adrian Kosmaczewski.

[3]https://mcfunley.com/choose-boring-technology
[4]https://moreeffectiveagile.com/