Issue 033: Management

Adrian Kosmaczewski

June 7th, 2021

Welcome to the thirty-third issue of *De Programmatica Ipsum*, dedicated to the subject of *Management*. In this edition:

- Graham analyzes the root causes of dysfunctional teams[1] and how to solve them.
- Adrian argues that the chasm between developers and managers[2] is hurting society as a whole, and it must end.
- In the Library section[3], Graham reviews Camille Fournier's book, *The Manager's Path*[4].

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 Maarten van den Heuvel[7] on Unsplash[8].

[1] https://deprogrammaticaipsum.com/on-the-absence-of-management/
[2] https://deprogrammaticaipsum.com/the-impossible-dialogue/
[3] https://deprogrammaticaipsum.com/category/library/
[4] https://deprogrammaticaipsum.com/camille-fournier/
[5] https://deprogrammaticaipsum.com/newsletter/
[6] https://deprogrammaticaipsum.com/contribute/
[7] https://unsplash.com/@mvdheuvel?utm_source=unsplash&utm_medium=referral&utm_content=creditCopyText
[8] https://unsplash.com/s/photos/management?utm_source=unsplash&utm_medium=referral&utm_content=creditCopyText

On The Absence Of Management

Graham Lee

June 7th, 2021

It is going to be hard to write anything about management for this month's issue, because the entire field is moribund. There has been no innovation in management for decades, supposed sea changes in thought have either failed to take hold or failed to deliver anything new, and we are all still mired in the industrialist, piecework thinking of Taylor's scientific management and workplace surveillance.

The acephalous nature of the modern workplace is evinced by the supreme slow pace of innovation in even the most high-tech of industries. The tablet computer was conceived in 1968, described in detail in 1972[1], and thanks to the breathtaking pace of private sector innovation was soon ready to pre-order...in 2010. Its place in a modern office, in a scalable suite of devices from wall-mounted smart TVs to pocketable badges, was clearly described[2] in a program that started in late 1987. We still do not have the product.

Meanwhile, the CEOs of the largest companies use their friendship with and gifts to their agents in the political system[3] to gain windfall tax breaks, meaning the poorest in society pay to keep that society together while they get the discounts. What do they do once they have dipped their scoop into the public purse? Do they increase wages? Hire more people from traditionally marginalised backgrounds? Fix worker abuses in their own offices and in their supply chain? Double down on R&D spend to create life-changing new categories of commodity?

[1] https://en.wikipedia.org/wiki/Dynabook
[2] http://www.cs.cmu.edu/~jasonh/courses/ubicomp-sp2007/papers/03-weiser-origins.pdf
[3] https://www.theverge.com/2021/1/20/22241646/apple-tim-cook-gave-the-first-mac-pro-to-trump

No, of course not, ha ha. How ridiculous. They engage in massive stock buybacks[4] that give the money directly to their investors, and go on patent purchasing sprees[5] to stop *anyone else* from accidentally innovating. Just to put the amount of money involved into perspective, that $90B stock buyback was enough money to buy the 87th largest company on the S&P 500 outright (Booking Holdings, at time of writing[6]), or the 15 smallest companies on the index (which are not the 15 smallest publicly traded companies, but the 486-501 *largest* taking Alphabet's two listings separately again).

Have the most valuable companies (i.e. the ones at the top of capitalism's high score table) done the most to improve standards of living, or introduced the most innovative and exciting products? Hardly. A quick scan of the top 11 (Alphabet is counted twice, because their class A and C stocks are listed separately) shows *one* genuinely life-changing company: Johnson & Johnson, one of the manufacturers of Covid-19 vaccines. Everyone else is in advertising (Alphabet, Facebook), finance (Berkshire Hathaway, Visa, JPMorgan Chase), retail (Amazon), reselling rechargeable batteries (Tesla), or renting you software that you used to buy (sorry, "license") outright (Microsoft, Apple). Read that again: fully six of the ten largest companies in the US are in the business of circulating capital, not producing anything: the ones who are producing anything are inventing new business models on old products to turn those into circulating-capital businesses too.

The latest product announced by the United States' largest company is a radio locator beacon (regularly in use since the 1950s), which adds to their portfolio of variously-sized television screens and wireless headphones. But of course it all uses the internet (developed in the 1960s with publicly-appropriated funding), which apparently legitimates the whole shebang being rented to you under the guise of "services".

So much for executive management and the C-suite, but what about middle management? What changes have their been to the way our employers interact with us software engineers in the 53 years since the job title was invented?

Once again, we see that Western culture's fascination with Adam Smith manufactory and Taylorian management science has held back innovation in this area, too. The old-school traditional software engineering approach featured managers trying to get more productivity (measured in function points[7] per day) from their over-stretched workforce. The new-school, post-Agile approach features managers trying to get more productivity from their over-stretched workforce, but now we measure in *story points per day* so that is progress, right?

This is not to say that the lightweight methods/Agile realignment has achieved nothing. Indeed, for many adopters it achieved a large subset of its goals: focusing on producing customer value instead of on fulfilling a predefined plan; frequent re-evaluation of practices and methods; integrating all development activities into a unified process rather than tackling them in distinct phases as isolated division-of-labour tasks. For some adopters, this transition did not go smoothly, so DevOps was born as a way to tell the same story using different words to try to get it to land again.

However, the entire concept of Agile software development required cultural shifts, and by and large those did not happen. Particularly, the people in the top and middle of the org

[4] https://www.nasdaq.com/articles/apple-results-soundly-beat-wall-street-targets-%2490-billion-buyback-announced-2021-04-28

[5] https://9to5mac.com/2019/03/04/lighthouse-patent-portfolio-apple/

[6] https://www.liberatedstocktrader.com/sp-500-companies/

[7] https://www.ifpug.org/publications-products/what-are-function-points-fact-sheet/

charts did not buy into the ideas, or see them as relevant. If agile is the thing that your engineering division does, or that your backend team does, then I am sorry, but I have two pieces of bad news for you. The first is that they are not "doing Agile", because they cannot. The second is that you are not managing your company, because you have failed to foster a shared cultural understanding of the work and the way of working.

The principles[8] behind Agile software development—but wait. We must point out that these are the principles that were behind the construction of the manifesto in 2001. This context is important. This is a document written in reaction to the world as it existed then, much as the "Rights of Man"[9] was written in 1791, or "Go To Statement Considered Harmful"[10] was written in 1968. Each of these is a reaction to the perception of the world its authors had at time of writing, and can only truly be understood through the lens of that time; and interpreted through the lens of our time.

Anyway, those principles include ideas that can be seen as much more collaborative, maybe even anarchistic, than could be found at many contemporary Anglo-American businesses.

> Business people and developers must work together daily throughout the project.

This does not *transcend* the division of labour—there are still "business people" and "developers" (you still hear engineers talking about "The Business" as if it is a separate organisation) but it does *mitigate* the division of labour. We go from analysts turning requirement-resources into specification-products, architects turning specification-resources into design-products, developers turning design-resources into code-products etc. into everyone working together to turn "customer needs", whatever they are, into "valuable software", whatever that is.

> Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.

This is the bit that requires a lot of trust from management. A *lot*. I mean, it is written right there: "trust them to get the job done". If you are demonstrating your distrust through presenteeism (related: "Management by Walking Around"), setting "stretch goals" above what your team forecasts it can achieve, or by ensuring that the daily stand-ups are actually daily status reports from your DRs, then it does not matter how nice a monitor you bought them or what the staff:foosball table ratio in your office is. You are still managing like it is the 19th century.

Here is a test of your level of organisational trust. Under what circumstances does the stand-up get canceled? If the answer is "when the manager cannot make it" or "when the product owner cannot make it", you are doing daily status reports. If the answer is "if we collectively decided we would not get any value from them", then there is some chance your managers trust you to get the job done.

> The best architectures, requirements, and designs emerge from self-organizing teams.

Once management trust its professionals to act professionally, it can let them get on with the task of apportioning the work. Now there is a true transcendence of Adam Smith, a genuine change in philosophy since the time of the pin factory and the innovation of paper money.

[8] https://agilemanifesto.org/principles.html
[9] https://en.wikipedia.org/wiki/Rights_of_Man
[10] https://homepages.cwi.nl/~storm/teaching/reader/Dijkstra68.pdf

We have gone full anarchism here: rather than trying to get management buy-in to some new methodology, the management are trying to get worker buy-in to the project they want completing.

Maybe that is the issue. Maybe it is "employment" that is the problem. Managers gonna manage, because they have paid for eight hours of your time per day and by golly gosh are they intent on getting out of those eight hours whatever it is that they can think of. Perhaps for software engineering management to get itself out of the moribund state it has been in for at least five decades, the managers need to be contracted by the workers.

Photo by Mathew Schwartz[11] on Unsplash[12].

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

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

The Impossible Dialogue

Adrian Kosmaczewski

June 7th, 2021

To a large extent, most of the software team managers I met in the course of my career were not able to judge neither the qualities nor the techniques required to create a software artifact. They were incapable of understanding the very work made by the teams I joined, the same work required to complete the software they clicked or touched at the end of each sprint. Yet most of those non-technical managers were by large the most inclined to take decisions that did have a major (usually negative) impact in said work, and not once, but rather very often.

Even worse, many were proud and even bragged about this fact.

On the other hand, the few tech-savvy managers I have had the fortune of working with took, paradoxically, few to almost no decisions at all; and when they did, they had such broad (positive) consequences, that they remain to this day engraved in my memory as examples to follow.

Anecdotes

This anecdotical experience is shared by nearly all software engineers I have met, regardless of nationality, culture, technology, or age. It is an important part of the folklore of writing code for a living. We all joke about this fact, day in, day out. We have all heard those stories. We tell those stories at conferences, we write books about them, we laugh at them at the bar.

Let us enumerate some of those classic stories[1].

[1] https://deprogrammaticaipsum.com/issue/issue-007-work/

A non-technical manager firing the politically-unaware yet gifted core of a software development team, while leaving in place those members who were not as versed in technical matters, yet excelled in political gambling, only to see the whole organization go bankrupt a few months later.

A company evaluating (and eventually firing) developers depending on the number of lines of code written, the number of tickets closed per unit of time, or any other metric easily visible in Jira, Redmine, GitLab, or similar tools.

A human resources department reminding developers, with passive-aggressive e-mails, that code must only be written within the office firewall from 9 AM to 5 PM, that home office[2] is a privilege with draconian constraints, that an impossible dress code must be respected during the hottest of summers in said office without ventilation, among other inane requirements.

A developer blocked from writing any code during months, because they could not install the required developer tools in their company-provided laptops, because security and rights and approvals and bureaucracy.

A brilliant asshole, usually a white man in his early thirties, dismissing months of work upon arrival, treating everyone as utter garbage, harassing[3] female employees, causing whole teams to quit en masse, yet receiving praise and bonuses by every manager, up to the CTO.

A team member suffering from mental health issues notifying via e-mail to their superior and HR department, both of whom never reply to said e-mail, and promptly dismiss the employment contract the following month without further due.

We could continue for pages, but that is not the point of this article. (For those asking, yes, these stories are all true, and sadly the author of these lines has witnessed them all.)

The Other Side

It is only fair to pay attention to both sides of the fence.

Most of the software developers I teamed with during my career were incapable of understanding the constraints and environments of the business paying them a salary. As a consequence, they were very eager to let the world know of their algorithmic considerations, put together in clumsily conclusions devoid of any actual applicability or substance, and more often that not, driven by the need to beef up their resumés. Sadly, this translated into expensive decisions, such as radical technology changes, complete rewrites, and many other catastrophes.

Even worse, many were proud and even bragged about this fact.

On the other hand, the few business-aware developers I have had the fortune of working with, had a growing interest in both commercial and software subjects, and were able to take technical decisions that actually benefited the bottom line of the organizations they worked for. They could enter in actual productive discussions with "the business" without the need for belittling anyone, engaging in respectful debate.

[2] https://deprogrammaticaipsum.com/issue/issue-030-home-office/
[3] https://deprogrammaticaipsum.com/issue/issue-006-diversity--inclusion/

Cost Management

The problem, as it turns out, is a modern version of C. P. Snow's observation in 1959 of The Two Cultures[4].

For the purpose of this discussion, this author will deliberately reduce the scope of management concerns to a single dimension: cost. In our capitalist world, sadly, the greatest (and often, the only) concern of "managers" is, without any doubt, the amount of money required to get a new "thing" online. Hence, here is the author hoping that the reader will endulge him in this diatribe.

Management holds dear to the phrase "you cannot manage what you cannot measure". Yet experts in a well-regarded magazine such as IEEE Computer are still undecided[5] on what to measure when it comes to software; and unsurprisingly, most root for counting single lines of code (the dreaded "SLOC") as a basic measure for productivity and advancement. If they do not know what else to measure, who can?

Well, maybe Barry Boehm[6] can. In case you have never heard of him, he is one of the first true researchers in the field of software engineering productivity and management.

In his 1981 paper titled "Software Engineering Economics"[7] he went as far as to propose a formula to evaluate the cost of software projects. You heard right; this is an algorithmic method for software cost estimation (you can rub your eyes, I will wait for you); that is, a mathematical formula that could have been implemented in Excel 35 years ago. And, lo and behold, SLOCs are at the basis of the whole equation.

This rather revolutionary contraption was called the COCOMO, or COnstructive COst MOdel. An unfortunate name, distorted 5 years later by the Beach Boys[8] as the soundtrack of a Hollywood blockbuster. But quite an incredible precedent, mostly forgotten by younger developers, and followed by the release of COCOMO 2.0[9] (by the same Barry Boehm) in 1995.

The major differences between both COCOMO models was the replacement of SLOCs as the only metric for software products, including "object points" and "unadjusted function points" as complements. The latter paper goes on to describe each one of these, and as you might infer, they were not immediately obvious to anyone outside of academia[10].

The first COCOMO, however, was algorithmically simple as to be included in a web service that some older readers of this magazine might remember: Ohloh[11]. Available from around 2005 to 2009, it provided an early form of "social network" around open source code. In the page of each project tracked by Ohloh there was a cost estimation based on the COCOMO model. This way you could learn that the Apache web server project cost millions of dollars, and took years of man work to be finished.

Simple, maybe not entirely effective or exact, but sadly completely forgotten.

---

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

[5] https://ieeexplore.ieee.org/document/7924249

[6] https://en.wikipedia.org/wiki/Barry_Boehm

[7] https://staff.emu.edu.tr/alexanderchefranov/Documents/CMPE412/Boehm1981%20COCOMO.pdf

[8] https://www.youtube.com/watch?v=fJWmbLS2_ec

[9] https://staff.emu.edu.tr/alexanderchefranov/Documents/CMPE412/Boehm1995%20COCOMO%202%20.pdf

[10] https://deprogrammaticaipsum.com/issue/issue-023-academia/

[11] https://en.wikipedia.org/wiki/Open_Hub

A decade later, Steve McConnell[12] tried to shine some light into the subject of cost estimation, mostly through his book "Software Estimation: Demystifying the Black Art"[13]. The biggest takeaway of this volume is the statement of the distinction between estimates, targets, and commitments[14]. Just because of this first chapter (freely available online), this book should become mandatory reading for both Computer Science graduates and MBA folks alike.

Maturity, Or Lack Thereof

Anthony Finkelstein from Imperial College wrote in 1992 about the "Software Immaturity Model"[15]. His paper bears a pun in its name, targeting the pompous Capability Maturity Model Integration[16] process, also known as CMMI, proposed in 1989 in a research report[17] sponsored by the United States Department of Defense. All very serious indeed.

But all very useless indeed as well. Merely three years later that report, Mr. Finkelstein realized what the rest of the industry took fifteen years more to figure out; that the CMMI model was incomplete, and that most software organizations required three more levels below the lowest CMMI classification. Hence the proposed "lunatic" (level -2), "stupid", and "foolish" (level 0) levels, to complete the CMMI-provided "initial" (1), "repeatable", "defined", "managed", and "optimising" (5) ones.

> The ad-hoc and chaotic processes followed by organisations at Level 1 can, by dint of exceptional individual and team effort, produce software. Level 0 foolish organisations act in such a way as to prevent this effort bearing any fruit. (...) Only by a miracle can a Level -2 organisation produce any useable software. As Level -2 organisations rarely get beyond specification they pin their hopes on automatically generating a program from that specification.

The truth is that most software organizations, unfortunately, are still better described using the 22 year-old allegory of The Big Ball of Mud[18].

A Way Forward

As stated previously in this article, we software practicioners "laugh at the bar" about these anecdotes. We must not. This is a serious problem, which has palpable effects in our society through the release of dysfunctional software, and it is the opinion of this author that this situation cannot continue any longer.

In spite of all of its merits, the whole Agile thing (and its most recent offspring, the famous DevOps[19]) is not enough to improve the current situation. You are not going to have better software products by valuing the items of the left more than those of the right. Those are good principles, but nothing else. Neither is a "correct application" (whatever that is) of Scrum, Kanban or Crystal going to solve this conundrum.

[12] https://deprogrammaticaipsum.com/steve-mcconnell/

[13] https://www.microsoftpressstore.com/store/software-estimation-demystifying-the-black-art-9780735605350

[14] https://www.microsoftpressstore.com/articles/article.aspx?p=2191414

[15] http://www0.cs.ucl.ac.uk/staff/A.Finkelstein/papers/immaturity.pdf

[16] https://en.wikipedia.org/wiki/Capability_Maturity_Model_Integration

[17] https://apps.dtic.mil/dtic/tr/fulltext/u2/a206573.pdf

[18] http://laputan.org/mud/

[19] https://deprogrammaticaipsum.com/issue/issue-016-devops/

I am not suggesting that developers should earn an MBA (although that could certainly be useful). However, reading books such as the following will help them understand the actual difficulties of running a business under real-world constraints: "The Ten Day MBA"[20] by Steven Silbiger, "The Personal MBA"[21] by Josh Kaufman, "Eric Sink on the Business of Software"[22] by Eric Sink, "Becoming a Technical Leader"[23] by Gerald Weinberg, "The Manager's Path" by Camille Fournier (reviewed by Graham[24] in this month's issue), and "Leading a Software Development Team"[25] by Richard Whitehead.

I am not suggesting managers should learn computer science (although that could certainly be useful). However, reading books such as "Geekonomics"[26] by David Rice, "The Phoenix Project"[27] by Gene Kim et al., "The Mythical Man-Month"[28] by Fred Brooks, "Peopleware"[29] by DeMarco & Lister, and "Agile"[30] by Bertrand Meyer will undoubtedly give them some useful perspective.

(Alternatively, you can follow this very publication, and maybe even contribute[31] to it, one objective of which is to build bridges between these two seemingly disparate worlds.)

The underlying message to software developers is this one: rolling your eyes in front of a business representative is *not* going to help anyone. Sit down, listen, and understand. Running a business can be as challenging as making that algorithm as efficient as possible, or to find a viable workaround to that stubborn, badly documented API from your vendor.

A similar message to business managers is: *do not* take "The IT Crowd"[32] as an example in management. The IT staff should not be relegated to the basement, because your "digital transformation" (whatever that is, and whatever shape it takes) requires you to have the "tech guys" on board, and to be taken into consideration.

The current impossible dialogue must become possible at all costs. This dialogue requires respect from both sides, a skill that cannot be acquired by reading books alone, however.

Let us not hold our breath for a revolution. So far, Sociocracy[33] and Holacracy[34] seem more like evolutions than revolutions in the field of management. The establishment is not yet ready to shake the status quo of how, when, by whom, and where work is to be performed yet, and will likely not be ready for change for quite a few decades, still.

That leaves us to think, with all pragmatism, that the best software in the years to come will be a byproduct of a healthy interaction between tech-savvy managers together with business-aware software engineers. The cross-pollination of both areas of expertise, by team members aware in both sides of the equation of the needs and constraints of the other, is the only formula we have at the moment for software teams to grow and improve.

[20] https://www.harpercollins.com/products/the-ten-day-mba-4th-ed-steven-a-silbiger?variant=32206243987490

[21] https://personalmba.com/

[22] https://deprogrammaticaipsum.com/eric-sink/

[23] https://leanpub.com/becomingatechnicalleader

[24] https://deprogrammaticaipsum.com/camille-fournier/

[25] https://www.amazon.com/Leading-Software-Development-Team-successfully/dp/0201675269

[26] https://www.amazon.com/Geekonomics-Real-Insecure-Software-paperback/dp/0321735978

[27] https://itrevolution.com/the-phoenix-project/

[28] https://www.amazon.com/Mythical-Man-Month-Software-Engineering-Anniversary/dp/0201835959

[29] https://www.amazon.com/Peopleware-Productive-Projects-Teams-3rd/dp/0321934113

[30] https://deprogrammaticaipsum.com/bertrand-meyer/

[31] https://deprogrammaticaipsum.com/contribute/

[32] https://en.wikipedia.org/wiki/The_IT_Crowd

[33] https://en.wikipedia.org/wiki/Sociocracy

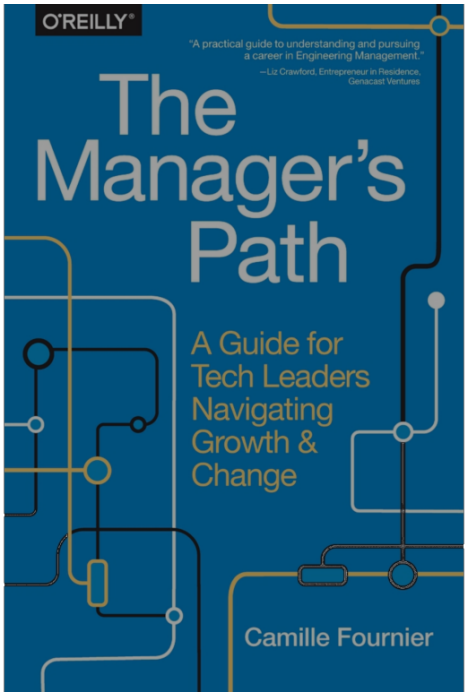
[34] https://en.wikipedia.org/wiki/Holacracy

Cover photo by the author.

Camille Fournier

Graham Lee

June 7th, 2021



As soon as Adrian and I agreed that Management would be the topic of this issue, I knew that I would share the benefits of Camille Fournier's book, *The Manager's Path*. It is the most succinct introduction to software engineering management for both managers and the managed out there.

Plenty has been written on the human aspects of software engineering, and some of it has even been read. Let us leave aside the human-computer interaction topics (design, UX, security etc), which are definitely well-covered. We will look at the other humans: those on our side of the IDE.

The typical names will come out when we list the classics of the humanity of computer programming. Fred Brooks is one such: what is the collection of essays in *The Mythical Man-Month* other than reflections on the failures of a successful engineering manager? Then there is Jerry Weinberg's *The Psychology of Computer Programming*[1], and *Peopleware: Productive Projects and Teams* by DeMarco and Lister. Books like *Code Complete*[2] or *Software Craftsmanship: the New Imperative* encourage us to reflect on our strengths and limitations as engineers, and of course the multitude of works on running Agile retrospectives and Scrum projects teach us how to walk the delicate balance that is managing a team that believes itself to be self-managing.

But Fournier's book is the only one I am aware of directly answering the Janus-faced question

[1] https://deprogrammaticaipsum.com/gerald-weinberg/
[2] https://deprogrammaticaipsum.com/steve-mcconnell/

of how to succeed at managing software engineers, and how to succeed at being managed as a software engineer. It is an important topic; many people know the truism that individual contributors (ICs) quit managers, they do not quit jobs. And most of us who become managers do so from the ranks of engineering, with no formal training and precious little support.

The thing with management is that it requires a completely different worldview than programming. Software is well suited to a positivist paradigm: you can understand how the computer works, and hypothesise that it will react to your stimulation in particular ways. When it does not react as you expect, then you have a problem, and you crack open the debugger. Management, on the other hand, is much more amenable to constructivist analysis: the dialogue between the managers and their reports shapes the mutual understanding of their relationship, but there is no objective "truth" to be determined.

That is why us programmers have such a difficult time when we first become managers. We learn simple rules—have a 1:1 every week; give your reports space on the 1:1 agenda; give praise publicly and criticism privately—and it seems like a simple enough algorithm to run. Then a report wants to move to a different country where we do not have an office, and another report is upset because they only got a great review when they expected an excellent review, and a third report who never brings up anything in 1:1s is pissed off because they were never given space to discuss their issues, and the whole thing goes to shit.

Fournier has been there. She has worked at small companies and large, in IC, middle, and senior management roles. She is not here to refine the algorithm given above, but instead to help us get comfortable with the idea that there is no algorithm, just heuristics. She describes the heuristics that have worked for her, as well as the times when they have not worked.

This book is useful even if you do not plan to be a manager. If you are an engineer and happy that way, then you are probably going to have to interact with managers, and knowing what they are thinking about and what they are trying to achieve will smooth that process. And becoming a more senior engineer means gaining influence, and that means the same skills that the managers are using. It is time to treat the Manager's Path.

Cover photo by the author.