Substack Archive

Table of Contents

Code security for software engineers

Nover

Stream the latest episode

Listen and watch now on [YouTube](), [Spotify](), and [Apple](). See the episode transcript at the top of this page, and timestamps for the episode at the bottom.

Brought to You by

?? **Statsig** ? - ? The unified platform for flags, analytics, experiments, and more. Statsig are helping make the first-ever Pragmatic Summit a reality. Join me and 400 other top engineers and leaders on 11 February, in San Francisco for a special one-day event. Reserve your spot here.

?? **Linear** ? - ? The system for modern product development. Engineering teams today move much faster, thanks to AI. Because of this, coordination increasingly becomes a problem. This is where Linear helps fast-moving teams stay focused. Check out Linear.

-

In this episode

As software engineers, what should we know about writing secure code?

Johannes Dahse is the VP of Code Security at Sonar and a security expert with 20 years of industry experience. In today's episode of *The Pragmatic Engineer,* he joins me to talk about what security teams actually do, what developers should own, and where real-world risk enters modern codebases.

We cover dependency risk, software composition analysis, CVEs, dynamic testing, and how everyday development practices affect security outcomes. Johannes also explains where AI meaningfully helps, where it introduces new failure modes, and why understanding the code you write and ship remains the most reliable defense.

If you build and ship software, this episode is a practical guide to thinking about code security under real-world engineering constraints.

Interesting quotes from the episode:

How code quality and code security are connected:

Gergely: "How does the quality of code relate to security?"

Johannes: "This is an underrated topic in the industry today. We talked about the null pointer exceptions or these slow regular expressions, right? That can lead to security issues. That's more of the obvious examples of bugs.

Think about unreadable code, not well-maintained code: that's often like spaghetti code. At first, it's not so obvious how this is connected to security.

If you think about how code is not easy to comprehend, not easy to review. Then you do pair programming or code reviews in your development team - then in that spaghetti code, you will more likely overlook security problems of your colleague.

Now, maybe someone found an issue or and issue and reports it back to you. As a developer, you have to fix it. But if it's not maintainable code, you cannot fix the security problem easily. So quality suddenly becomes a security issue in how the attacker window stays open longer!

Your code quality is super related to code security, especially now with AI-generated code. We typically see poor quality of code here. And that becomes a problem for security."

How AI is introducing new security issues:

Gergely: "How do you think AI is changing code security?"

Johannes: "We see a change in how code and applications are built. Traditionally, you had this backend/frontend split.

In the backend, you had a database. Remove that database, then you don't have a SQL injection risk anymore.

As soon as you add an LLM to the backend, you have to deal with prompt injection vulnerabilities. The attacker can modify the system prompt or manage do some prompt engineering and then mess with the LLM's logic or the output. It's taking time both for the attackers to adjust to this, and the industry to adjust in defending against it."

Gergely: "What about with coding assistance? Are you seeing things change in terms of how we think about code security?"

Johannes: "**The big problem [with AI coding assistance] in terms of security is that you produce code much more faster.** Writing code is not the challenge anymore. Suddenly, the new bottleneck is how are you verifying all that code, right? And if you don't verify it:that leads to security issues or quality issues. Quality issues, on the long run, lead to security problems."

On software composition analysis:

Gergely: **"What is software composition analysis (SCA)?"**

Johannes: "It's a technique where we look at manifest files and your list of dependencies, depending on the package manager you use. This list of dependencies is checked against a database of known security problems. Those are called the CVEs. Then you can - with software composition - map, for example, that this specific Log4j version of your library is vulnerable to the Log4 shell vulnerability that is known. And then it can warn you."

The Pragmatic Engineer deepdives relevant for this episode

What is Security Engineering?

Mishandled security vulnerability in Next.js

Okta Schooled on Its Security Practices

Timestamps

(00:00) Intro

(02:31) What is penetration testing?

(06:23) Who owns code security: devs or security teams?

(14:42) What is code security?

(17:10) Code security basics for devs

(21:35) Advanced security challenges

(24:36) SCA testing

(25:26) The CVE Program

(29:39) The State of Code Security report

(32:02) Code quality vs security

(35:20) Dev machines as a security vulnerability

(37:29) Common security tools

(42:50) Dynamic security tools

(45:01) AI security reviews: what are the limits?

(47:51) AI-generated code risks

(49:21) More code: more vulnerabilities

(51:44) AI's impact on code security

(58:32) Common misconceptions of the security industry

(1:03:05) When is security "good enough?"

(1:05:40) Johannes's favorite programming language

References

Where to find Johannes Dahse:

? LinkedIn: https://www.linkedin.com/in/johannes-dahse-112b3057

Mentions during the episode:

? Sonar: https://www.sonarsource.com

? State of Code Security Report by Sonar https://www.sonarsource.com/resources/the-state-of-code-security-report/

? OWASP Top Ten: https://owasp.org/www-project-top-ten

? Software Composition Analysis: https://en.wikipedia.org/wiki/Software_composition_analysis

? CVE Program: https://www.cve.org

? SAST: https://en.wikipedia.org/wiki/Static_application_security_testing

? What is DAST: https://github.com/resources/articles/what-is-dast

? Stack Overflow AI survey: https://survey.stackoverflow.co/2025/ai

? Go: https://go.dev

? Java: https://www.java.com

-

Production and marketing by Pen Name.

Holiday gift ideas for techies

Nover

It's that time of the year: the Black Friday and Cyber Monday sales are on, and the annual festive marketing blitz is just around the corner - or already underway. It makes now a good time to start thinking about gifts, but techies can be a tough crowd for this, as we often already own the practical things we need.

In order to help you give gifts which are actually wanted this year, Elin, of this publication, and I have put together a list of ideas in this article. Alongside personal recommendations, we've also crowdsourced recommendations from fellow techies on X, Bluesky, and Threads, covering:

Office accessories

Computer add-ons

Health and well-being

Gadgets

Gaming and games

Travel & wearables

Books and stationery

Kitchen goodies

Many products listed below are currently discounted in the sales, and the Pragmatic Engineer is also offering a very special Black Friday / Cyber Monday deal for annual subscriptions. Claim it here until Monday.

As always, none of the links below are affiliates (meaning I make no money from purchases), and I've not been paid to mention any product or category. See my ethics statement for more.

For more recommendations, see our holiday gift guide from 2023 and book recommendations from 2021.

Programming note: this week, we'll have a podcast episode tomorrow (Wednesday), and no edition of The Pulse on Thursday. Regular programming resumes next week after Thanksgiving.

1. Office accessories

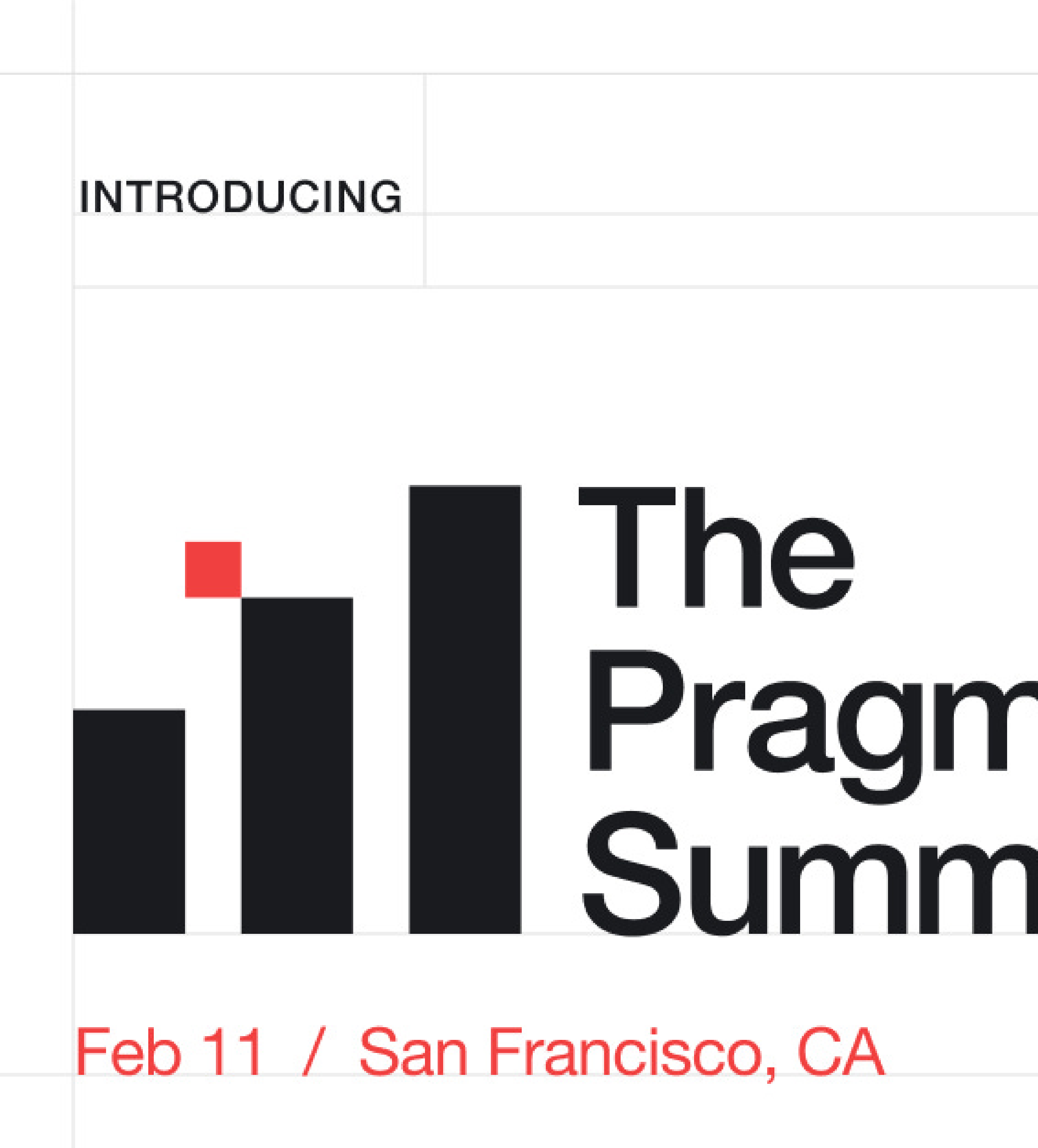
Ember Temperature Control Mug - keeps coffee or tea warm, even when it's forgotten about because you're focused on coding or other tasks. A hot drink stays drinkable for up to 1.5 hours, thanks to its built-in battery that charges on a nifty wireless-charging coaster.

(Content could not be rendered due to complex HTML formatting)

The Pulse #154: Cloudflare takes down half the internet - but shares a great postmortem

Nover

Before we start: **The Pragmatic Summit** has officially launched, and you can find more details - and the first few speakers announced - on **the summit's website**.



11 February 2026, in San Francisco - I hope to see many of you there! You can also **apply here directly.**

How AI will change software engineering - with Martin Fowler

Nover

Stream the latest episode

Listen and watch now on YouTube, Spotify, and Apple. See the episode transcript at the top of this page, and timestamps for the episode at the bottom.

Brought to You by

?? **Statsig** ? - ? The unified platform for flags, analytics, experiments, and more. AI-accelerated development isn't just about shipping faster: it's about measuring whether, what you ship, actually delivers value. This is where modern experimentation with Statsig comes in. Check it out.

?? **Linear** ? - ? The system for modern product development. I had a jaw-dropping experience when I dropped in for the weekly "Quality Wednesdays" meeting at Linear. Every week, every dev fixes at least one quality isse, large or small. Even if it's one pixel misalignment, like this one. I've yet to see a team obsess this much about quality. Read more about how Linear does Quality Wednesdays - it's fascinating!

-

In this episode

Martin Fowler is one of the most influential people within software architecture, and the broader tech industry. He is the Chief Scientist at Thoughtworks and the author of *Refactoring* and *Patterns of Enterprise Application Architecture*, and several other books. He has spent decades shaping how engineers think about design, architecture, and process, and regularly publishes on his blog, MartinFowler.com.

In this episode, we discuss how AI is changing software development: the shift from deterministic to non-deterministic coding; where generative models help with legacy code; and the narrow but useful cases for vibe coding. Martin explains why LLM output must be tested rigorously, why refactoring is more important than ever, and how combining AI tools with deterministic techniques may be what engineering teams need.

We also revisit the origins of the Agile Manifesto and talk about why, despite rapid changes in tooling and workflows, the skills that make a great engineer remain largely unchanged.

Interesting quotes from the episode

On what non-determinism introduced by LLMs will mean for software engineering:

Gergely: "Is this the first time we're seeing a tool that is so wide to certain software engineering that is non-deterministic?"

Martin: "It's a whole new way of thinking. It's got some interesting parallels to other forms of engineering.

In other forms of engineering, you think in terms of tolerances. My wife's a structural engineer. She always thinks in terms of what are the tolerances, how much extra stuff do I have to do beyond what the math tells me because I need it for tolerances. Because I mostly know what the properties of wood or concrete or steel are, but I've got to go for the worst case.

We need probably some of that kind of thinking ourselves.

What are the tolerances of the non-determinism that we have to deal with? We need to realize that we can't skate too close to the edge because otherwise we're going to have some bridges collapsing. And I suspect we're going to do that, particularly on the security side.

We're going to have some noticeable crashes. I fear because people have got skated way too close to the edge in terms of the non-determinism of the tools they're using.

On how many big companies are "complicated messes," because of humans.

Martin: "I remember chatting with somebody who had joined an established bank. They joined from a startup and one of their jobs was to modernize the bank.

Their comment was: 'now that I've been here three years, I think I can *understand* the problem. I've got some idea of what I can do, what can be done. But it just takes you that long to just really understand where you are in this new landscape because it's big and it's been around a long time.'

Companies are complicated - and often not logical because they are built by humans, not by computers. And there's all sorts of history in there because all sorts of things happened because so-and-so met so-and-so and had it around with so-and-so, and all of these things kind of percolate over time. And this vendor came in here, and that was popular over here, and then the person who liked this vendor got moved to a different part of the organization. Somebody else came in who wanted a different vendor.

And all of this stuff builds up over time to a complicated mess. And any big company is going to have that kind of complicated mess**.** It's very hard to not get that situation."

The Pragmatic Engineer deepdives relevant for this episode

Vibe coding as a software engineer

The AI Engineering stack

AI Engineering in the real world

What changed in 50 years of computing

Timestamps

(00:00) Intro

(01:50) How Martin got into software engineering

(07:48) Joining Thoughtworks

(10:07) The Thoughtworks Technology Radar

(16:45) From Assembly to high-level languages

(25:08) Non-determinism

(33:38) Vibe coding

(39:22) StackOverflow vs. coding with AI

(43:25) Importance of testing with LLMs

(50:45) LLMs for enterprise software

(56:38) Why Martin wrote Refactoring

(1:02:15) Why refactoring is so relevant today

(1:06:10) Using LLMs with deterministic tools

(1:07:36) Patterns of Enterprise Application Architecture

(1:18:26) The Agile Manifesto

(1:28:35) How Martin learns about AI

(1:34:58) Advice for junior engineers

(1:37:44) The state of the tech industry today

(1:42:40) Rapid fire round

References

Where to find Martin Fowler:

? X: https://x.com/martinfowler

? LinkedIn: https://www.linkedin.com/in/martin-fowler-com

? Website: https://martinfowler.com

Mentions during the episode:

? The code migration tool mentioned in the episode: OpenRewrite
https://www.thoughtworks.com/radar/tools/openrewrite

? UK Atomic Energy Authority: https://www.gov.uk/government/organisations/uk-atomic-energy-authority

? Books by Jim Odell: https://www.amazon.com/stores/James-J.-Odell/author/B001HMPDVG

? Thoughtworks: https://www.thoughtworks.com

? TDD, AI agents and coding with Kent Beck:
https://newsletter.pragmaticengineer.com/p/tdd-ai-agents-and-coding-with-kent

? Extreme Programming: https://martinfowler.com/bliki/ExtremeProgramming.html

? Software architecture with Grady Booch:
https://newsletter.pragmaticengineer.com/p/software-architecture-with-grady-booch

? Thoughtworks Technology Radar: https://www.thoughtworks.com/radar

? Rebecca Parsons on LinkedIn: https://www.linkedin.com/in/dr-rebecca-parsons

? Conversation: LLMs and Building Abstractions: https://martinfowler.com/articles/convo-llm-abstractions.html

? James Lewis on LinkedIn: https://www.linkedin.com/in/james-lewis-microservices

? Cursor: https://cursor.com

? Resharper: https://www.jetbrains.com/resharper

? The Learning Loop and LLMs: https://martinfowler.com/articles/llm-learning-loop.html

? Godot: https://godotengine.org

? Stackoverflow: https://stackoverflow.com

? Inside Linear's Engineering Culture: https://newsletter.pragmaticengineer.com/p/linear

? Waterfall model: https://en.wikipedia.org/wiki/Waterfall_model

? *Refactoring: Improving the Design of Existing Code*:
https://www.amazon.com/Refactoring-Improving-Existing-Addison-Wesley-Signature/dp/0134757599

? Ralph Johnson on LinkedIn: https://www.linkedin.com/in/ralphejohnson

? John Brant on LinkedIn: https://www.linkedin.com/in/john-brant-80b87056

? Don Roberts: https://refactory.com/don-roberts

? Adam Tornhill's website: https://www.adamtornhill.com

? *Patterns of Enterprise Application Architecture*:
https://www.amazon.com/Patterns-Enterprise-Application-Architecture-Martin/dp/0321127420

? *Patterns of Distributed Systems*: https://www.amazon.com/gp/product/0138221987

? Jim Highsmith on LinkedIn: https://www.linkedin.com/in/jhighsmith

? Bob Martin on LinkedIn: https://www.linkedin.com/in/robert-martin-7395b0

? How Claude Code is built: https://newsletter.pragmaticengineer.com/p/how-claude-code-is-built

? Birgitta Böckeler on LinkedIn: https://www.linkedin.com/in/birgittaboeckeler

? Simon Willison's Weblog: https://simonwillison.net

? Applying Domain-Driven Design and Patterns: With Examples in C# and .NET:
https://www.amazon.com/Applying-Domain-Driven-Design-Patterns-Examples-ebook/dp/B0054KOKQQ

? Expert Generalists: https://martinfowler.com/articles/expert-generalist.html

? Ruby: https://www.ruby-lang.org

? Smalltalk: https://en.wikipedia.org/wiki/Smalltalk

? *Thinking Fast and Slow*: https://www.amazon.com/Thinking-Fast-Slow-Daniel-Kahneman/dp/0374533555

? *The Power Broker: Robert Moses and the Fall of New York*:
https://www.amazon.com/Power-Broker-Robert-Moses-Fall/dp/0394720245

? *The Path to Power (The Years of Lyndon Johnson, Volume 1)*:
https://www.amazon.com/Path-Power-Years-Lyndon-Johnson/dp/0679729453

? *Concordia*: https://boardgamegeek.com/boardgame/124361/concordia

-

Production and marketing by Pen Name.

Career paths for software engineers at large tech companies

Nover

Across tech, the average tenure of software engineers seems to be rising, not least in Big Tech where it has increased rapidly. With today's chilly job market having a dampening effect on the number of engineers switching jobs, it's possible that staying in a role for years will become pretty normal for many in our industry.

So, if you can see yourself at your current workplace for the longer term, it's sensible to consider the career path options available, and how to get promoted to the next level.

To shed light on these topics and others, I sought out someone who has managed large engineering orgs at a massive company. **Ethan Evans** is precisely such a person; he was a Vice President of Engineering at Amazon, and has overseen the growth and promotions of more than 1,000 engineers(!) over the course of his career.

Now recently retired from the online retail giant, Ethan can candidly discuss how large companies operate, and which tactics and strategies *really* work and help colleagues to enjoy thriving careers in big workplaces.

These days, Ethan teaches engineers and engineering managers how to get promoted faster, and runs live courses. He's also built a 24/7 personalized "AI career coach", and shares career growth advice in his weekly newsletter, Level Up. His class on fast career growth currently has a 25% discount for the US Thanksgiving holiday, from today through 2 Dec - *you can check it out here.*

In this issue, we cover:

Good performance as a mid-level engineer. Execute independently and try not to complain too much.

How to get a slam-dunk promotion to Senior. Have big ideas that are also correct, solve problems leadership didn't know existed, see around corners, and more.

Tactics to get promoted to Senior. Agree a plan with your manager and be realistic about how long it takes.

Getting promoted to Principal Engineer. Standout technical expertise, tackling ambiguous & large-scale problems, and paying attention to the business, are all key.

Should you be a manager? As you grow more senior, there may be opportunities to become a manager. But is it the best path for you?

When to switch to management. Switching early (at L5) or later (L7+) is rare, and is most common at senior level (L6).

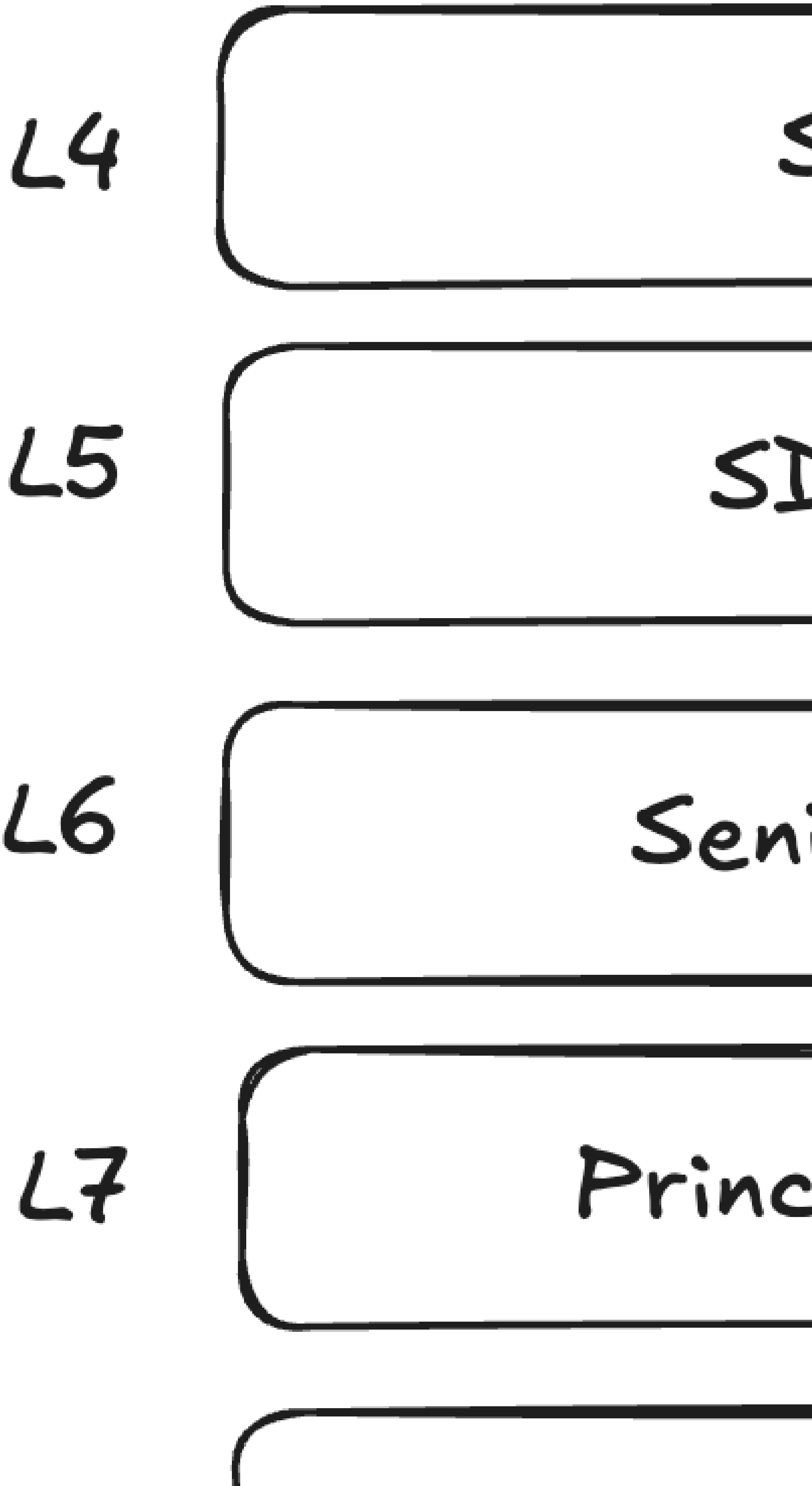
Different management levels. How being a manager at L5, L6, and L7 levels differ.

Throughout this article, we use Amazon's career levels (**bolded** in the bulletpoint list below) which are a bit different from most places (listed alongside for comparison). Find out more about levels and how they compare at Levels.fyi

L5: mid-level engineer. This is L4 at places like Google, Meta, and Uber

L6: senior. At many companies, this is the L5 level

L7: principal. After the senior level at places like Google is the Staff Engineer level (L6), then the Senior Staff level (L7) and then Principal level (usually L8).


L4
S
L5
SD
L6
Seni
L7
Princ

Amazon's Software Development Engineer (SDE) career levels. L9 is unique to Amazon

Find out more about being a software engineer at the online retail giant in our deepdive Inside Amazon's engineering culture.

With that, it's over to Ethan:

What do the career path choices for a mid-level engineer with between 2 and 6 years' experience look like from the point of view of a Vice President managing hundreds of engineers? In my career as an Amazon VP, I oversaw the growth and promotion of over 1,000 engineers, and saw some grow to senior engineers, and then even on to the Principal or Staff levels. Some people switch to management quickly, while others do this deeper into their careers.

The "correct" path for you is subjective and depends on your goals and environment. What I will do in this article is give some insights into these paths from a higher perspective, and reveal the choices which leaders make about colleagues and promotions. My goal is to help you choose the right path and navigate it successfully.

First, some information about the shorthand used at Amazon and Google; this can be applied to other companies on the website, levels.fyi. A mid-level engineer (SDE-2), is level 5, or L5, at those companies. The first level for managers is also L5. Senior engineers (SDE-3s) are L6, as are more experienced small group managers. Principal or Staff Engineers are L7, as are Senior Managers (managers of managers, with 25 to 80 people in their teams).

It's easier and shorter to type "L5", etc., so I'll use these terms while going through the "three paths" available to a mid-level engineer (L5) who wants to progress. These are:

Grow as an IC, first to L6, then perhaps to L7 and beyond

Switch to management early, as an L5

Become a senior engineer (L6), then switch to management

1. Good performance as a mid-level engineer. (L5 at Amazon)

When thinking about career paths, we have to begin with what "very good" performance looks like. Until you are seen as a very good L5 engineer, you won't be thought of as Senior Engineer or manager material. So, from the perspective of the VP level, what makes someone stand out as a good L5 who might be ready for more?

First, independent execution. A good L5 can take a reasonable design and just build it. They can figure out some missing details on their own, and don't need daily or even weekly guidance. They know their craft and can get the work done, while knowing when to raise problems and ask for help.

Limited complaining. Leaders might tolerate a high-maintenance engineer for a while, but will never consider a habitual complainer as a candidate for progression. It's fine to push back on ideas, to ask questions, and to disagree. However, when a decision is made, leaders need people who just go do it - even when they think there's a better way. Similarly, all jobs have unpleasant sides; for many engineers, this is the Ops part of DevOps: fixing bugs, migrating services, and being oncall. Leaders are going to be reluctant to promote those who only want the "good" work of new design and development. Why? Because *all* the work needs to be done!

Before you can move up to L6 or management, it's necessary to be seen as a valuable, effective member of your team. After all, why would anyone promote someone who doesn't contribute meaningfully and is hard to work with?

If you've been told you can be difficult, but you nonetheless believe your talent gives you a pass on changing your approach - well, it doesn't. Remember, your Director or VP has hundreds of engineers on their team and also has a quota for "unregretted attrition". Placing difficult colleagues in that quota works just fine if they are pains to work with. *Note from Gergely: we cover more on Amazon's unregretted attrition target and its infamous "PIP culture" in Inside Amazon's engineering culture.*

2. Traits for a slam-dunk promotion to Senior (L6 at Amazon)

Once you are a solid L5, what are the key factors that make a VP see you as an L6 in waiting?

Even if there are a couple of layers between you and your VP, good executives know their top engineers. I was rarely involved in selecting L5s for potential promotion once I became a VP, but was almost always part of the approval process and was in the room when promotions were discussed, as part of efforts to keep the bar consistent across my whole organization.

Here are the features of a slam-dunk promotion case from L5 to L6:

Big ideas that are also *right*. Proposing unexpected ideas that work and are valuable to the team's mission. We expect L6's to design, contribute to the architecture, and invent things. Showing you can do more than just building what you're told is the next big step.

An antipattern: endless pet projects that are exciting for the engineer, like learning a new language or tool, but which don't actually contribute to their team's needs.

Significant independent initiative (the "L6 scope" project). Leaders want to see you can handle a bigger challenge, solo. Can you work with product management, collaborate with them on a design that meets the product goals, and then implement that large design over the course of months of consistent work? Exactly what each company and leader looks for will vary, but usually it's something like designing, building, and launching a new service with interesting complexity. It can also mean thoroughly refactoring an older, complex service that's outdated and hard to maintain, but still vital.

Speed tip: it's often easier to get the chance to take on an ugly refactoring than it is to find some completely new need. It may be quicker to move up by showing your seniority and helpfulness here, than to wait to innovate something "new."

Solving problems the leadership didn't know existed. A winning recipe for impressing leadership is almost always to spot and address a big problem we didn't know we had. It shows judgment, initiative, and a desire to help. These are all things we want more of, so they're rewarded.

Seeing around corners: highlighting and solving looming problems currently out of sight to your manager, pointing things out before they become problems, and volunteering to prevent them, is also highly prized.

Become the "go-to" person: It stands out to develop unique technical expertise, and be someone whom others on the team refer to; particularly if that includes a higher-level team member. Nothing says "ready for L6" better than a few L6s saying they turn to you on certain matters.

Mentoring and developing others. Teams have constant turnover and new hires, and managers can't mentor new engineers ourselves for lack of time. So, we value those who teach and share skills.

Antipattern: "Don't bother me, I'm a coder!" Like avoiding oncall and other work, people who are unhelpful by claiming they only focus on development, won't move up.

Able to influence - or lead - others. Often an extension of mentorship; being able to win over others to your ideas, or to direct the efforts of entry-level team members to get something done, is a sign of maturity. Every senior person is expected to be able to organize, direct, and manage complex work beyond their own personal tasks. While managers may do the hiring, firing, performance reviews, etc., it's necessary for senior ICs to be able to get others to work with them on bigger goals.

Antipattern: being personally difficult to work with is the opposite of bringing people together.

Operational excellence. Being the person who doesn't just take their turn with oncall, but who instead dives in, finds root causes, and builds tools that make the system more reliable. Some engineers avoid operational work, so we value and respect those who willingly do it thoroughly.

Crisis management. Being the engineer who jumps on the call when there is a weekend outage, or who volunteers to go onsite with an angry customer. Such engineers show they understand the needs of the business, and that we can count on them when needed most.

Antipattern: absent, blames others, complains.

You don't need to do all of these things to become an L6; but the more you do, the better.

3. Tactics to get promoted to Senior

Given the list of things you can do to be seen as ready for L6, how should you go about it?

Agree a plan with your manager. Ideally, discuss it with your manager and agree a course of action. While in theory your manager should be thinking about your development, a wise engineer never leaves this to chance. By talking to your manager, you make your growth goal clear and get the specifics of what they expect. Note that if they have concerns about your current performance as an L5, you are going to need to fix those first.

Do the work while building a positive relationship. Many engineers dislike hearing that relationships matter because they believe standards should be objective and that "opinions" shouldn't matter. Yet, so much depends on "soft judgments": from who gets the chance to take on new work, to whether or not a project is really big enough to be L6 work. Remember that your manager probably began as an engineer, too. They are just as good as you at arguing why a project is easy and trivial, or complex.

Some engineers also hear "sucking up" when I mention relationships, but this is not what I mean. A better way to look at it is simply as being pleasant and helpful. Remember, you are probably not the only L5 on the team; a manager is more likely to help those who are helpful and friendly, in the same way as you might help someone you like instead of a person who's difficult and annoying.

Be grounded about how long it takes. Once you and your manager agree a plan, realize that promotion will probably take 1 to 2 years from when you are a solid L5. There are many ways to show you can do well, and while you don't need to do everything on the list above, you are very unlikely to complete enough of them in less than a year.

How to get promoted faster

Some people struggle to accept any kind of timeframe. In their minds, the instant they show mastery of some skills, then promotion should follow. But speaking as a leader, it's not that simple. While you may believe you have shown mastery, from the outside it is often hard to tell if someone is actually good, or just got lucky on the first try. We want to avoid mistakes, so prefer to see a few things delivered in order to be sure that it is skill, not good fortune.

That said, the way to move as fast as possible is to talk to your manager, have a clear plan, and then start cranking through it. You cannot know when a new person will be hired for you to mentor, or when a big outage will happen. You will progress fastest by being flexible about which elements you demonstrate and by jumping on opportunities as they arise. It's fine to decline a project if the timing is bad, but then don't be surprised if your promotion takes longer than the person who grabbed it.

4. Getting promoted to Principal or Staff Engineer (from L6 to L7)

Now we've covered L5 to L6, let's discuss L6 to L7. The same basics apply: first, be seen as a good, reliable L6 who is solid at that level. Then, share your goal with your manager, get a plan, etc. Below are the key elements of a slam-dunk promotion to Principal / Staff (L7):

The Pulse #153: Is Microsoft too early to agentic OS - like with smartphones?

Nover

The Pulse is a series covering events, insights, and trends within Big Tech and startups. Notice an interesting event or trend? Hit reply and share it with me.

Today, we cover:

Microsoft too early to agentic OS - like with Windows phones? The tech giant wants Windows to be an "agentic OS", but developers hate the idea and could move to MacOS or Linux. Is Microsoft in the process of repeating its Windows Mobile 6 error?

Industry pulse. Agents becoming the default way to write code, more AI-coding startups adopt metered pricing, Cursor the highest-valued dev tools company, 1-day onboarding at AI-native companies, and more.

Inside Cursor's unique engineering culture. A completely different way of hiring, rapid shipping as the norm, many former founders at the company, and more.

Before we start: as a paid newsletter subscriber, you can apply a few days early to **The Pragmatic Summit**!

I shared a teaser on Tuesday: and I'm now very excited to share the first major in-person The Pragmatic Engineer event. This will be a gathering of 400 top engineers and leaders in San Francisco on **11 February 2026** for a day of practical insights, high-signal sessions, and peer exchange focused on building great products and scaling world-class engineering teams.

Netflix's Engineering Culture

Nover

Stream the latest episode

Listen and watch now on [YouTube](), [Spotify](), and [Apple](). See the episode transcript at the top of this page, and timestamps for the episode at the bottom.

Brought to You by

?? **Statsig** ? - ? The unified platform for flags, analytics, experiments, and more. Statsig enables two cultures at once: continuous shipping *and* experimentation. Companies like Notion went from single-digit experiments per quarter to over 300 experiments with Statsig. Start using Statsig with a generous free tier, and a $50K startup program.

?? **Linear** ? - ? The system for modern product development. When most companies hit real scale, they start to slow down, and are faced with "process debt." This often hits software engineers the most. Companies switch to Linear to hit a hard reset on this process debt - ones like Scale cut their bug resolution in half after the switch. Check out Linear's migration guide.

-

In this episode

What's it like to work as a software engineer inside one of the world's biggest streaming companies?

In this special episode recorded at Netflix's headquarters in Los Gatos, I sit down with Elizabeth Stone, Netflix's Chief Technology Officer. Before becoming CTO, Elizabeth led data and insights at Netflix and was VP of Science at Lyft. She brings a rare mix of technical depth, product thinking, and people leadership.

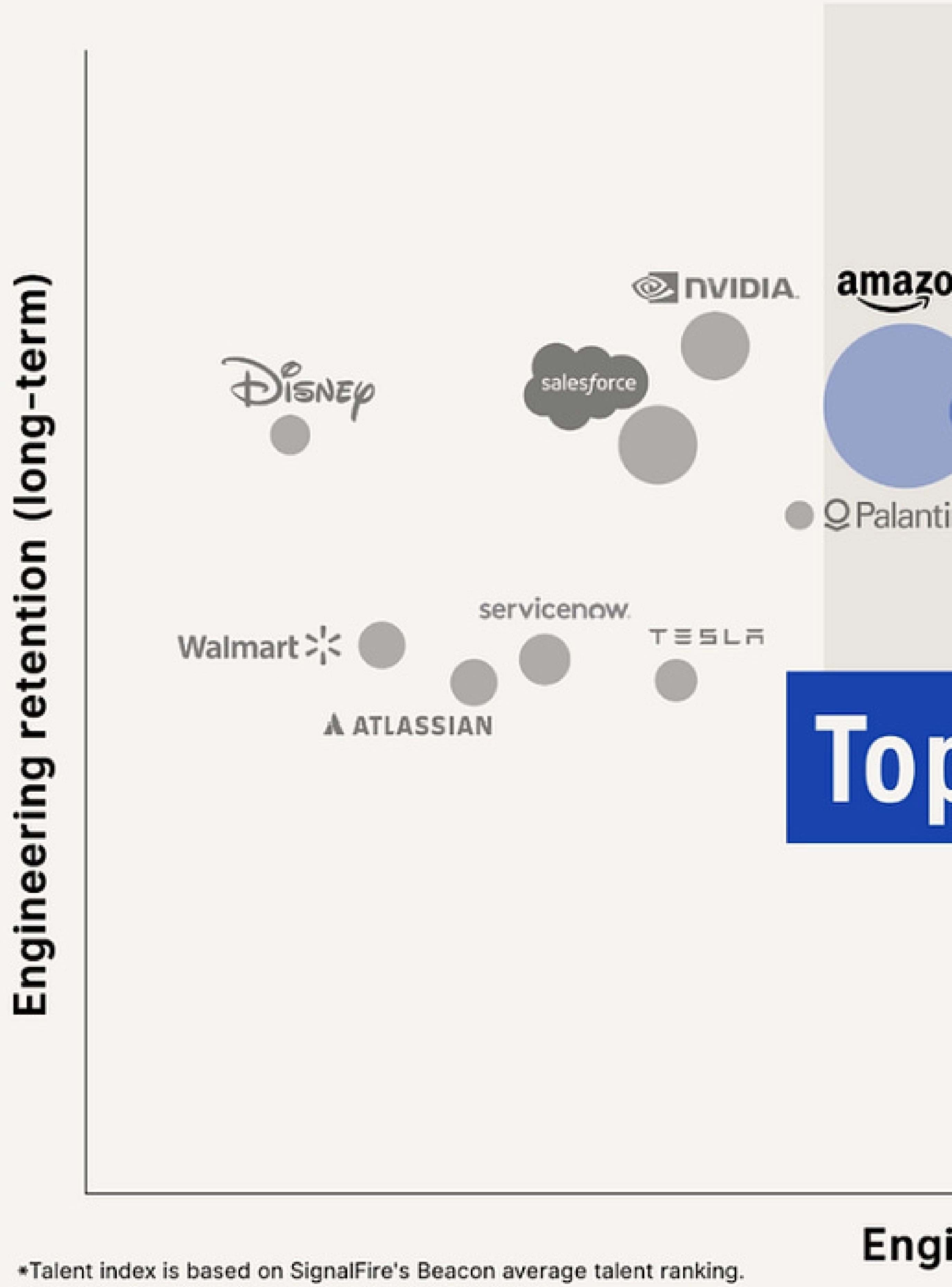
We discuss what it means to be "unusually responsible" at Netflix, how engineers make decisions without layers of approval, and how the company balances autonomy with guardrails for high-stakes projects like Netflix Live. Elizabeth shares how teams self-reflect and learn from outages and failures, why Netflix doesn't do formal performance reviews, and what new grads bring to a company known for hiring experienced engineers.

This episode offers a rare inside look at how Netflix engineers build, learn, and lead at a global scale.

Interesting details about Netflix

A "top" company for engineering talent and retention. From this report by Signal Fire, Netflix is in the far right corner of this chart, meaning the company hires strong engineering talent, and retention is above industry average:

Cracking the code: Orgs that



*Talent index is based on SignalFire's Beacon average talent ranking.

Source: SignalFire

Hiring of new grads. Netflix has only hired senior software engineers for the first 25 years of the company - up to 2023, when the company introduced engineering levels. Netflix now hires new grads: and because they started from 0% new grad ratio, they have a lot more space to play with. *Tech companies hiring more interns and new grads is a recent trend we've observed.*

No formal performance review process. At most tech companies, the annual or bi-annual perf process is heavyweight, and takes away easily a month of focus from engineering managers, and many engineers as well. Netflix doesn't have a heavyweight process: instead, they put continuous feedback in place, and a few lightweight check-ins, including the Keeper Test. They also have an annual 360 review process. This is used as a "safety net," and sounds like a lightweight feedback process, meant to highlight issues that continuous feedback would not catch.

Areas where AI tools work well for Netflix - these are these:

Prototyping

Documenting code

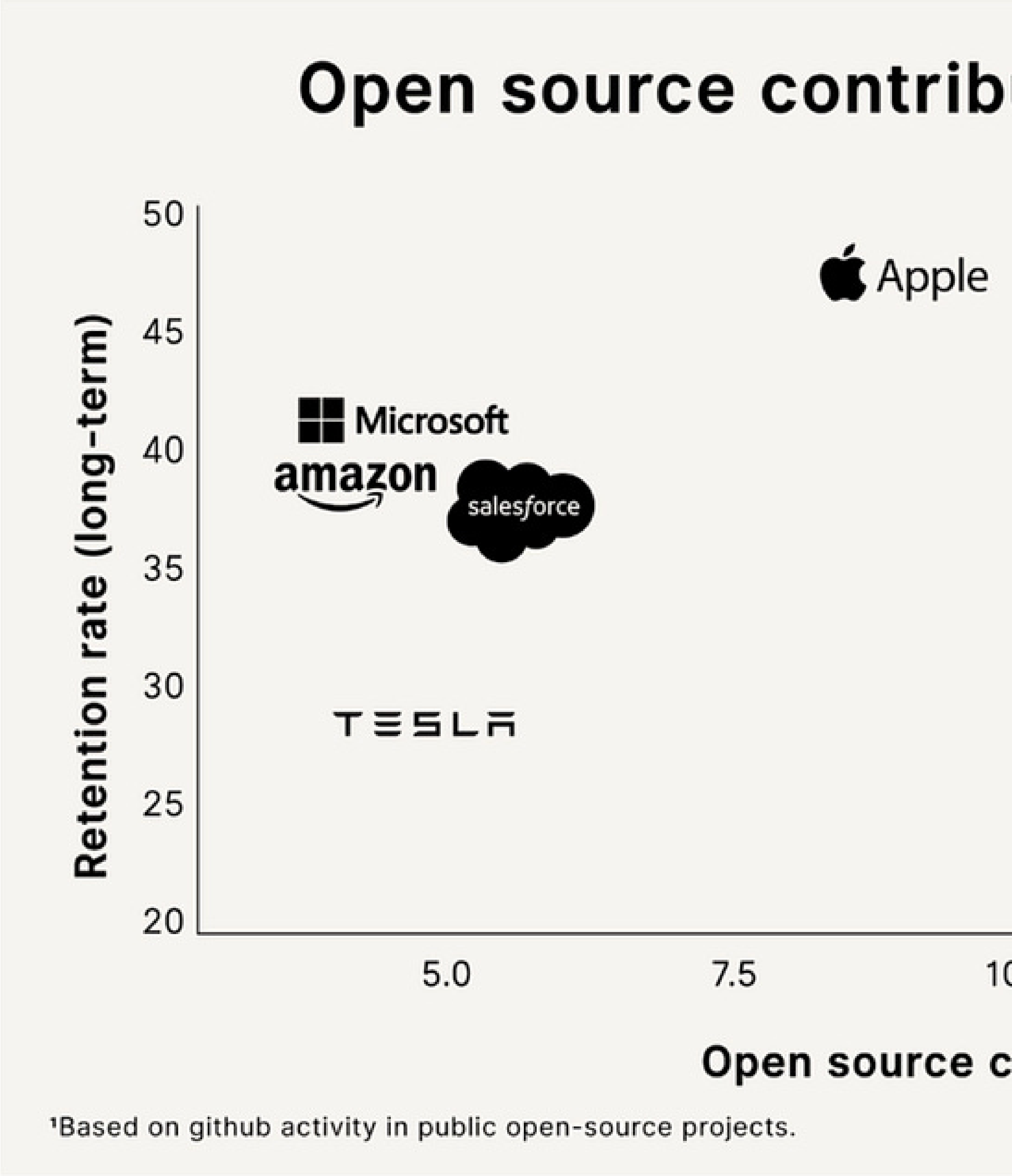
Working on large migrations

Detecting issues (anomaly detection and root cause analysis)

Elizabeth emphasized how they don't see AI as a "silver bullet:" they experiment with tools, and have observed them to have come a long way in usefulness, from a few years ago.

Unexpectedly heavy open source investment. Netflix has won 9 Emmy awards for open source contributions - mostly for video steaming contributions.

Across all of Big Tech, Netflix has the highest percentage of all engineers contribute to open source: about 20% of them. This was a surprising fact for me to learn, and we went into details in the podcast:



About 1 in 5 engineers at Netflix contribute to open source. Source: SignalFire

The Pragmatic Engineer deepdives relevant for this episode

The end of the senior-only level at Netflix

Netflix revamps its compensation philosophy

Live streaming at world-record scale with Ashutosh Agrawal

Shipping to production

What is good software architecture?

Timestamps

(00:00) Intro

(01:44) The scale of Netflix

(03:31) Production software stack

(05:20) Engineering challenges in production

(06:38) How the Open Connect delivery network works

(08:30) From pitch to play

(11:31) How Netflix enables engineers to make decisions

(13:26) Building Netflix Live for global sports

(16:25) Learnings from Paul vs. Tyson for NFL Live

(17:47) Inside the control room

(20:35) What being unusually responsible looks like

(24:15) Balancing team autonomy with guardrails for Live

(30:55) The high talent bar and introduction of levels at Netflix

(36:01) The Keeper Test

(41:27) Why engineers leave or stay

(44:27) How AI tools are used at Netflix

(47:54) AI's highest-impact use cases

(50:20) What new grads add and why senior talent still matters

(53:25) Open source at Netflix

(57:07) Elizabeth's parting advice for new engineers to succeed at Netflix

References

Where to find Elizabeth Stone:

? LinkedIn: https://www.linkedin.com/in/elizabeth-stone-608a754/

Mentions during the episode:

? Bringing the Best in VFX and Virtual Production Together as Eyeline: https://about.netflix.com/en/news/bringing-the-best-in-vfx-and-virtual-production-together-as-eyeline

? Open Connect: https://openconnect.netflix.com

? Jake Paul vs. Mike Tyson Was the Most Streamed Sporting Event in History: https://www.netflix.com/tudum/articles/jake-paul-vs-mike-tyson-live-release-date-news

? Live from Netflix, It's a New Chris Rock Stand-Up Special: https://www.netflix.com/tudum/articles/chris-rock-live-standup-special

? What is Chaos Monkey: https://www.techtarget.com/whatis/definition/Chaos-Monkey

? The Scoop: Netflix's historic introduction of levels for software engineers - exclusive: https://blog.pragmaticengineer.com/netflix-levels

? Code, culture, and competitive edge: Who's winning the engineering talent game?: https://www.signalfire.com/blog/report-engineering-talent-retention

? Alliance for Open Media: https://aomedia.org

? Behind the Streams: Three Years Of Live at Netflix. Part 1: https://netflixtechblog.com/behind-the-streams-live-at-netflix-part-1-d23f917c2f40

-

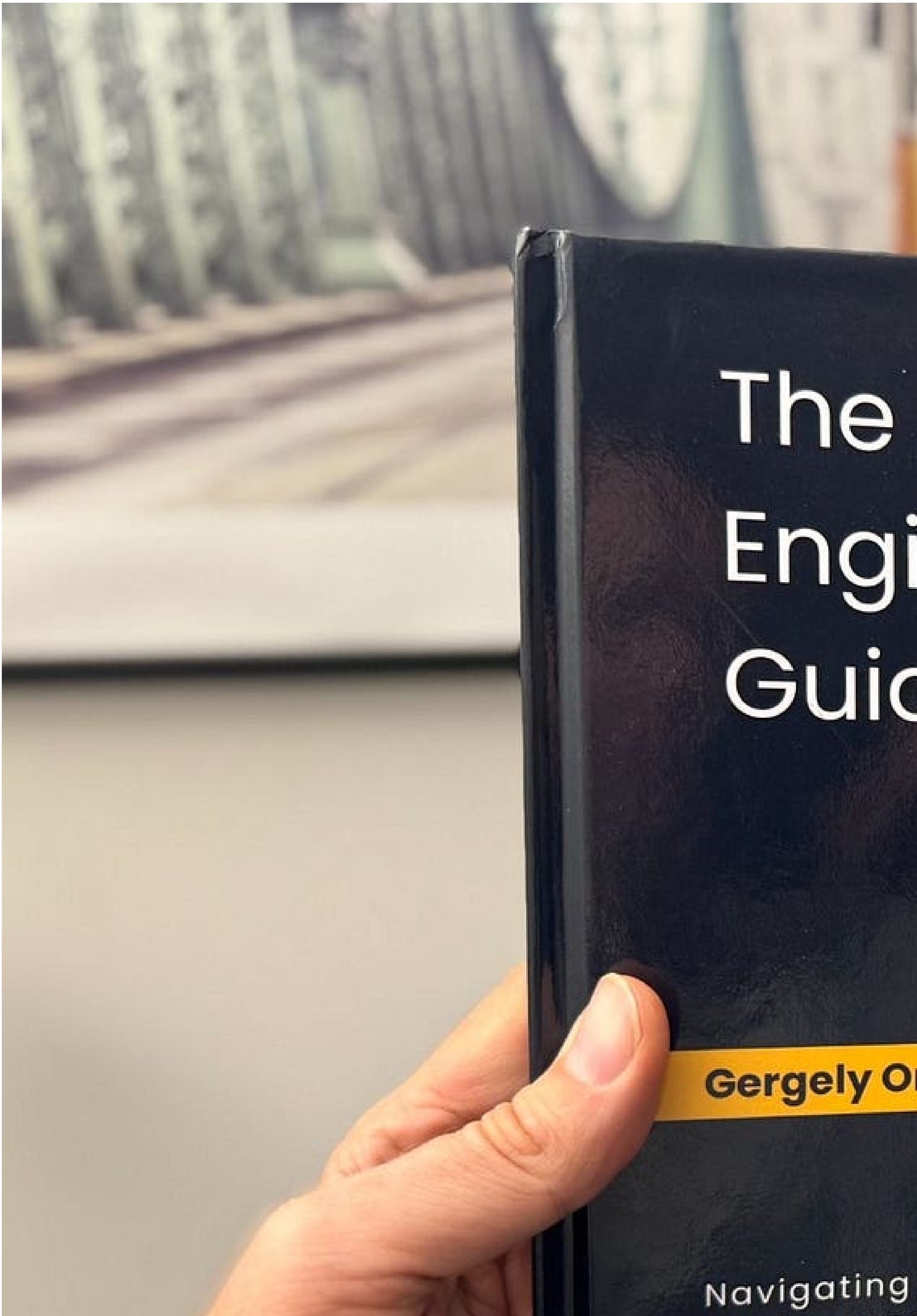
Production and marketing by Pen Name.

The Software Engineer's Guidebook: a recap

Nover

Before we start: I'll be in San Francisco on 11 Feb, 2026. I'm working on something special for engineers and engineering leaders. I can't wait to share more. 11 February - save the date!

Two years ago almost to the day, I published The Software Engineer's Guidebook. Originally, it came out in paperback, then as an ebook and an audiobook. I'm happy to share that it is now available as a hardcover, and is also on the O'Reilly platform.


The
Engi
Guid
Gergely Or
Navigating

Hardcover edition: Durable and could make a nice gift for the holidays. You can get it here

Today, I'd like to draw back the curtain on the process of writing a book as a techie:

Pitching to publishers. And why I ended up breaking up with a publisher.

Self-publishing. The tools I used for writing and the platforms I published on.

Traveling to Mongolia to meet the startup which translated the book. A 30-person startup called Nasha Tech translated the book for the benefit of their company and the Mongolian tech ecosystem.

How much did my book earn? $611,911 in two years from royalties on 40,000 copies sold, to date. We need more good books in tech, so I hope that sharing these numbers inspires other techies to write them.

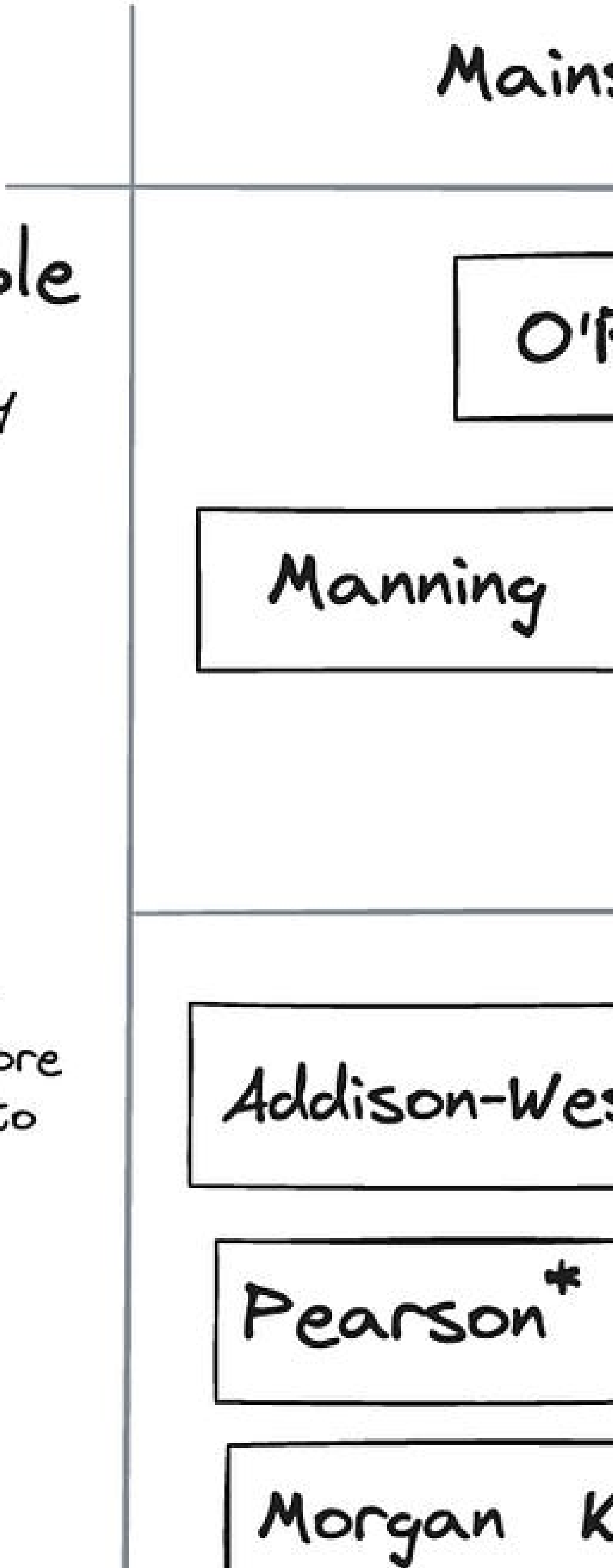
Learnings from writing my book. It's hard to judge a book's impact, but good books stay valuable for longer - that's why they're hard to write.

1. Pitching to publishers

In late 2019, I was an engineering manager at Uber, the ridesharing app. For the first time in my career, I was a manager of managers: suddenly, I had "skip level" software engineers, and it was here that the inspiration came to write a book that provides some advice and observations for these tech professionals.

During a 1:1 catchup with a new joiner skip-level engineer, they asked about getting up to speed in the workplace faster. They were at the Software Engineer 2 (L4), and wanted to figure out how to get to the Senior engineer level (L5A at Uber). I thought it would've been nice to give them a book with a bunch of pointers about what it takes to become an effective software engineer in this environment.

With that inspiration to write a book about professional growth at large tech companies and startups, I looked for publishers who could help make it a reality. I pitched my book to 3 publishers behind the highest quality tech books in the industry - O'Reilly, The Pragmatic Bookshelf, and Manning. O'Reilly and The Pragmatic Bookshelf passed, but Manning said yes.



Mains

Highly reputable

Guaranteed quality

O'R

Manning

Seemingly fewer titles than before and/or harder to pitch to

Addison-Wes

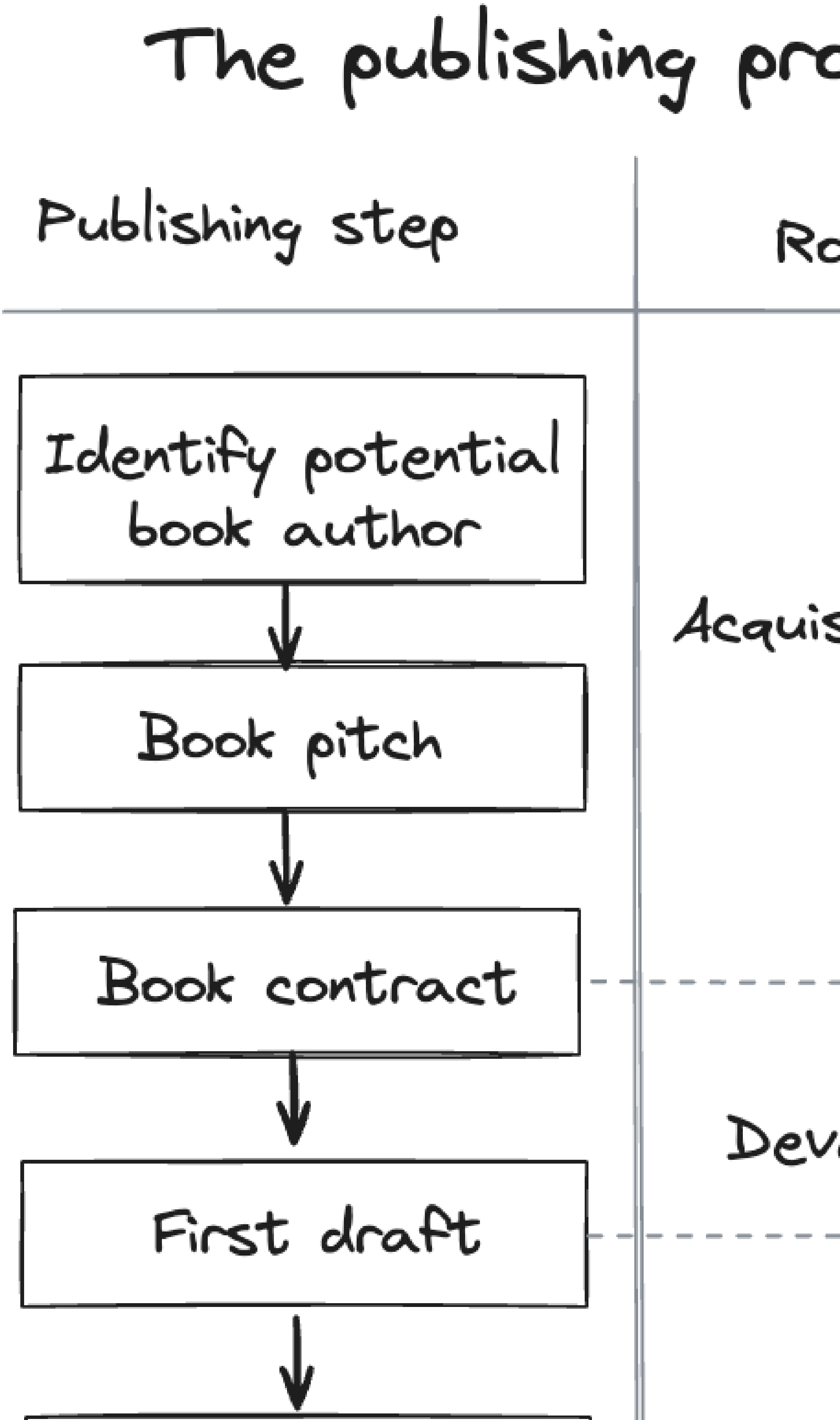
Pearson*

Morgan K

Mental model of publishers in 2025. I pitched my book to the 3 most reputable tech publishers

I learned a lot by working with a publisher: the process of preparing a book to be ready for sale is much more involved than I'd assumed! I even came to appreciate why publishers may keep up to 85-90% of sales revenue, with authors typically getting 7-15% of royalties from print sales.

The chart below sets out the publication process at a typical tech book publisher, from the author pitching their idea for a book, all the way through to its launch:


The publishing pro
Publishing step
Ro
Identify potential book author
Book pitch
Acquis
Book contract
Dev
First draft

Steps to publish a book with a publisher

Unfortunately, after three months with Manning and one editorial review, it was clear that the publisher and my book weren't a good match. At the time, Manning had a very opinionated template for creating books which I felt pushed my own in a direction I wasn't happy with: I felt the book would be dumbed down, and that following some of the guidance would weaken it. Also, I was less comfortable with some of the conditions and feared my hands could be tied if I wanted to share draft material online,. Also, I would not be able to choose the book title. So, we parted ways. You live and learn!

I shared more details about my experience of working with publishers in this article, including the financials of choosing a publisher as a writer.

2. Self-publishing

The biggest downside of not having a publisher is the lack of deadline pressure. Below is the rough structure I used for writing my book:

Getting the idea and the motivation

Rough outline

Procrastination about the topic

Table of contents

Writing the draft - the longest part of all

Copyediting

Content feedback

Final copyedit

More procrastination about the final draft

Production layout editing: getting the book ready for print

Cover design

Test printing

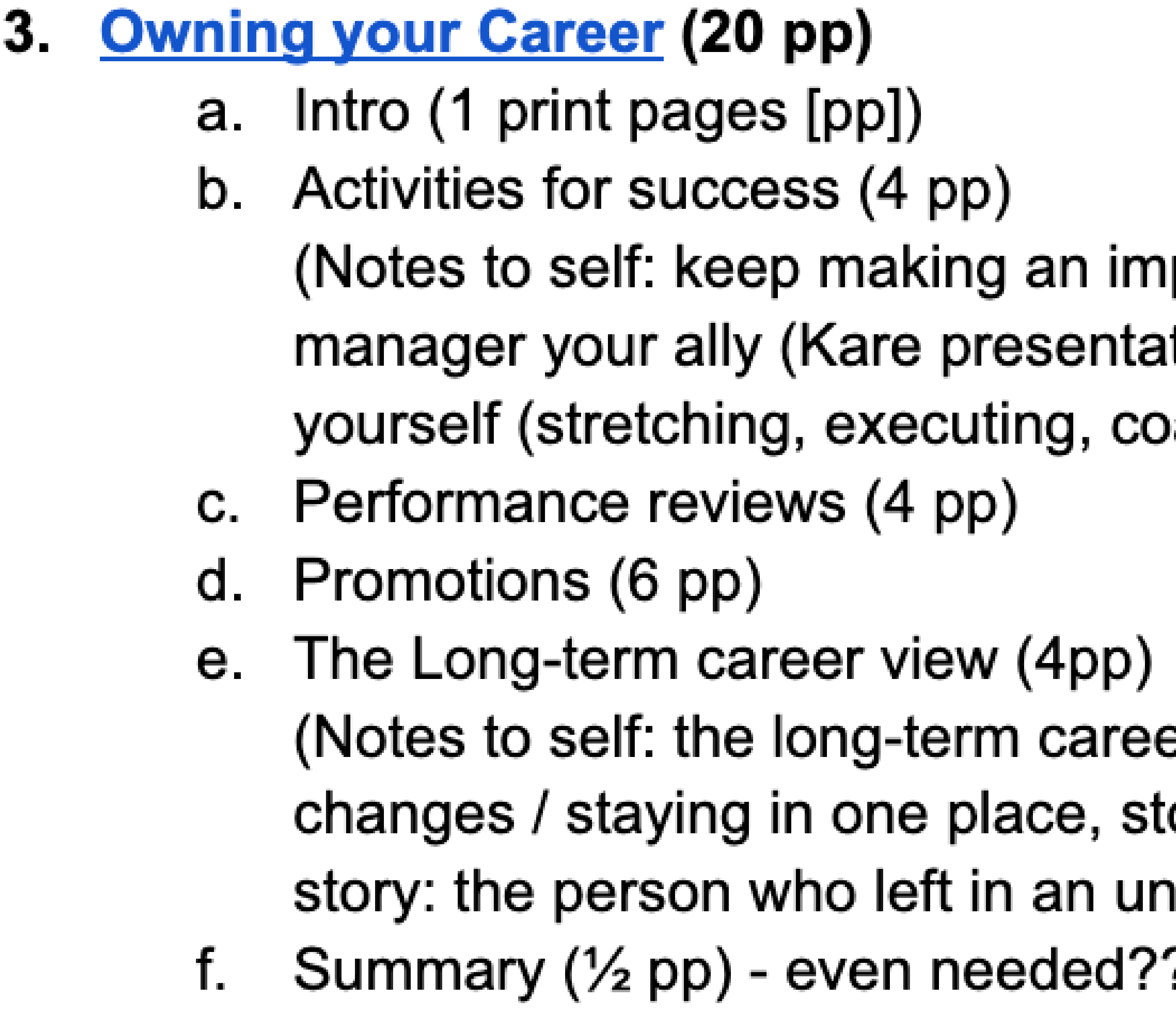
Launch planning

Launch!

Weighted table of contents

One very useful thing I took from Manning's process is to start the book with a table of contents, where you estimate the rough number of pages one section will be about. This is helpful because the published version should not be too long or too short, so this helps set its scope and how topics should be tackled.

Here is an example of the weighted table of contents for one chapter:



3. Owning your Career (20 pp)
   a. Intro (1 print pages [pp])
   b. Activities for success (4 pp)
   (Notes to self: keep making an im
   manager your ally (Kare presenta
   yourself (stretching, executing, co
   c. Performance reviews (4 pp)
   d. Promotions (6 pp)
   e. The Long-term career view (4pp)
   (Notes to self: the long-term care
   changes / staying in one place, st
   story: the person who left in an un
   f. Summary (½ pp) - even needed?

My weighted table of contents

This exercise resembled estimating the size of a piece of work during the planning for a software project, and provided a sense of what the longer and shorter parts of the book would be.

Write, write, write

The biggest part of writing a book is... writing it. I tried to have some regular cadence of writing, but struggled to get into one. I ended up making progress in bursts, and shared drafts on social media - like this one on healthy and unhealthy teams. This generated more ideas and I continued writing.

It's easy to get distracted and procrastinate. A 400-page book is too large a project to take on in one go, and sometimes parts of the book change shape: an estimated 4 pages on something can turn into 20 pages a week later. It was like treading water: not making much progress with the book as a whole and being distracted by interesting topics.

Starting The Pragmatic Engineer Newsletter to finish the book

Two years into writing the book, its completion was forever 6 months away. I realized my writing process, with no publisher involved, was inefficient and unfocused.

In August 2021, I started The Pragmatic Engineer Newsletter with the hope that writing a weekly newsletter would speed up completion of the book. My thinking was to send out a draft version of a part of the book as a newsletter every few weeks.

The idea was sound, but I quickly realized that online newsletter issues can be a lot more in-depth than a chapter in a book can be, due to the issue of space constraints. Also, newsletters are timely by nature (covering what's happening, *right now*), whereas a book often covers ideas that remain relevant for years.

I am thankful to have started The Pragmatic Engineer Newsletter, but doing so didn't hasten the publication of The Software Engineer's Guidebook. It actually delayed it by another two years.

Tools I used for self-publishing

There's many tools available for writing a book, not including new AI tools for generating ideas - or even doing the writing in certain cases. Here's the tools I chose:

Writing process: Google Docs and Craft. Most of my drafts were in Google Docs, and all my ideas and notes went into Craft.

Creating an ebook: Vellum. A one-off purchase, which helps generate e-book and print versions. I didn't use it for print generation, as the software seems to be optimized for fiction.

Print layout: Overleaf. The PDF layout for print took significant effort, as I wanted to have an index in the end of the book. I ended up hiring a LateX expert who helped create a nice print layout, which would have been challenging for me to do, and time-consuming to learn.

Reader feedback: Betabooks. I wasn't entirely satisfied as it's pretty janky to use. In hindsight, sharing drafts in Google Docs and letting readers add comments might have been equally effective.

ISBN numbers: when self publishing, you need to purchase your own ISBN and these depend on your region. ISBN Services is a popular choice for the US. I'm residing in the Netherlands, so used the Dutch Mijn ISBN.

Book cover design: Canva. Lots of templates to work with, and it's easy to work with the exact size requirements for print covers, as well. The cover you see today is my wife's work.

Publishing platforms

I publish my books on these platforms:

Print book: Amazon Kindle Direct Publishing (KDP) and Ingram Spark. Amazon covers most countries - but not all; notable gaps include India. Ingram Spark allows libraries and bookshops to order the book. If your title is not expected to be highly popular, Amazon KDP will probably be more than sufficient for print distribution.

Ebook: Gumroad (to sell the ebook DRM-free), Amazon KDP, Kobo Store, Google Play, and PublishDrive. Publishdrive is a neat service that publishes ebooks and audiobooks to all large and small platforms. In hindsight, I'd probably just use them to publish to all major platforms.

Audiobook: Voices (formerly: Findaway Voices). For some strange reason, Amazon's audiobook platform (ACX) only allows residents of the US, Canada, UK, and Ireland, to publish on it. Being a resident in the Netherlands, I had to go with a third-party platform to publish to Amazon. Voices used to be owned by Spotify, and has been spun out into its own company. If I published today, I'd likely just go with PublishDrive for my audiobook, as well.

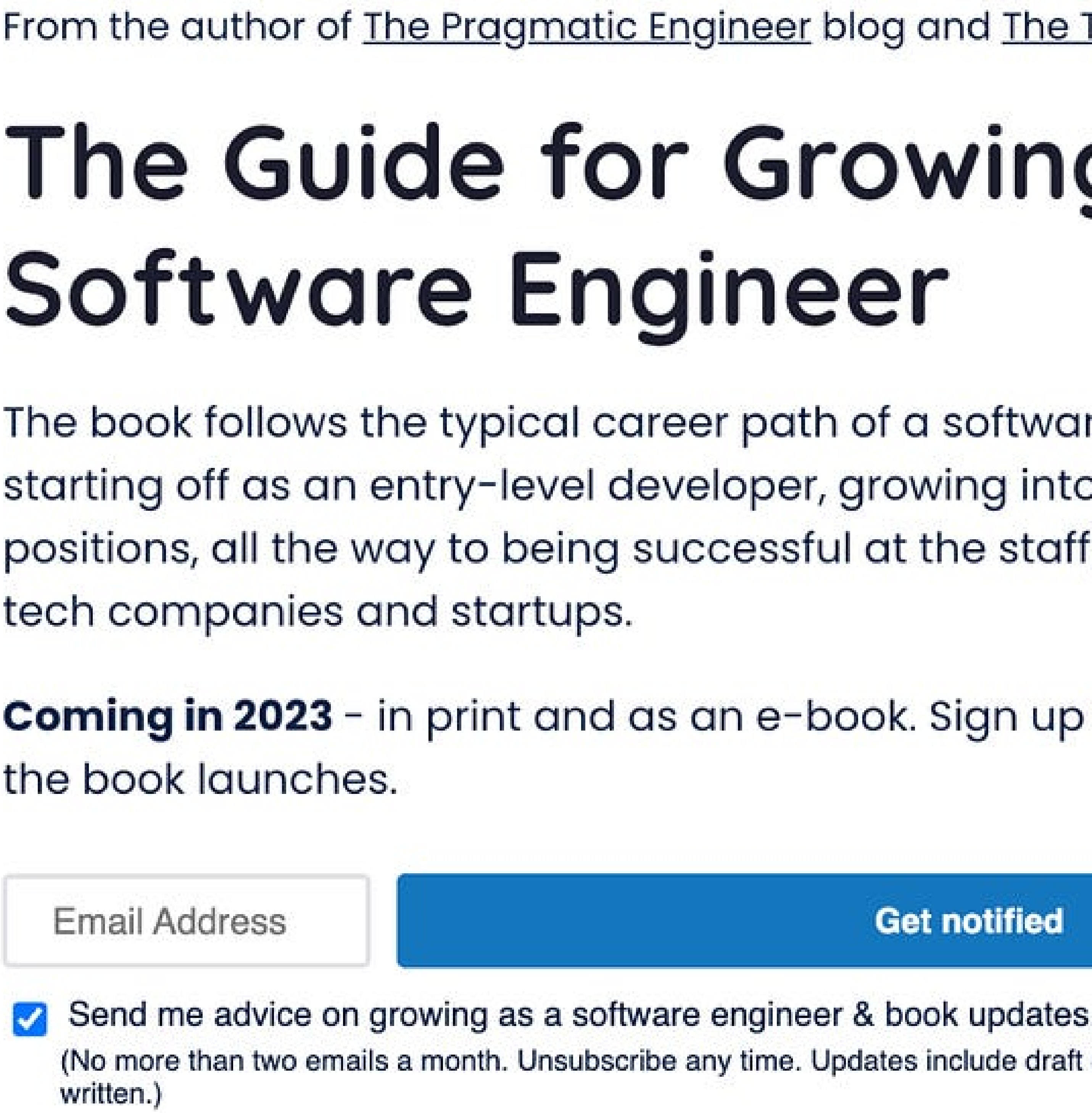
There's a long tail of ebook and audiobook platforms, and it makes sense to use an aggregator service that publishes to most, if not all of them, and which then sends a combined payout, instead of lots of small payments from each platform.

Don't forget the launch

Going with a publisher has the benefit that they coordinate launch activities, and also list your book on their New Releases pages, which leads to added awareness. Superior publishers also reach out to potential reviewers, and are hands-on in helping with the launch.

I knew I couldn't count on launch support, so built a landing page where I collected emails of people who wanted to know when the book was released. To incentivize this, I provided details about the book, and the table of contents.

In hindsight, this was one of the best things I did: a good chunk of sales at launch came from people who received the email that the book was ready for purchase. Here's how the book's website looked, back then.



From the author of The Pragmatic Engineer blog and The T

The Guide for Growing Software Engineer

The book follows the typical career path of a softwar starting off as an entry-level developer, growing intc positions, all the way to being successful at the staff tech companies and startups.

Coming in 2023 - in print and as an e-book. Sign up the book launches.

☑ Send me advice on growing as a software engineer & book updates
(No more than two emails a month. Unsubscribe any time. Updates include draft written.)

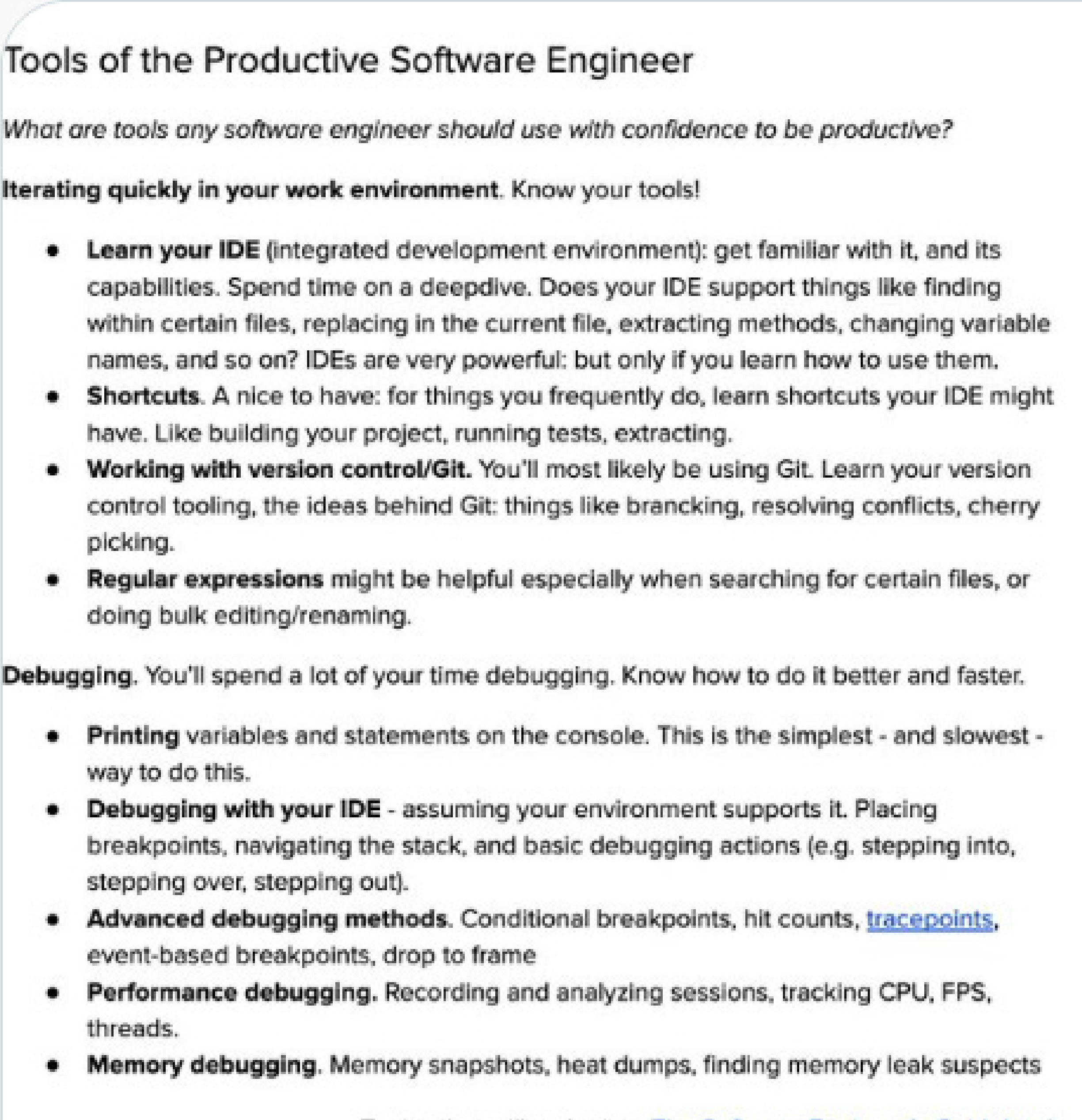
Landing page for the book, pre-publication

Another approach that worked well was having a dedicated social media account for the book, where I shared drafts while working on certain chapters. I was surprised by how many reshares a page or two of the work-in-progress book got, like this one.

The Software Engineer's Guideb...

What are tools of a productive softwa

- Knowing your IDE and its capabilitie
- Debugging. Know how to use tools t
- Automated tests. Learn to write robu
- CI/CD
- Coding workflow. Small changes > b



Tools of the Productive Software Engineer

What are tools any software engineer should use with confidence to be productive?

Iterating quickly in your work environment. Know your tools!

- **Learn your IDE** (integrated development environment): get familiar with it, and its capabilities. Spend time on a deepdive. Does your IDE support things like finding within certain files, replacing in the current file, extracting methods, changing variable names, and so on? IDEs are very powerful: but only if you learn how to use them.
- **Shortcuts**. A nice to have: for things you frequently do, learn shortcuts your IDE might have. Like building your project, running tests, extracting.
- **Working with version control/Git.** You'll most likely be using Git. Learn your version control tooling, the ideas behind Git: things like brancking, resolving conflicts, cherry picking.
- **Regular expressions** might be helpful especially when searching for certain files, or doing bulk editing/renaming.

Debugging. You'll spend a lot of your time debugging. Know how to do it better and faster.

- **Printing** variables and statements on the console. This is the simplest - and slowest - way to do this.
- **Debugging with your IDE** - assuming your environment supports it. Placing breakpoints, navigating the stack, and basic debugging actions (e.g. stepping into, stepping over, stepping out).
- **Advanced debugging methods**. Conditional breakpoints, hit counts, tracepoints, event-based breakpoints, drop to frame
- **Performance debugging.** Recording and analyzing sessions, tracking CPU, FPS, threads.
- **Memory debugging**. Memory snapshots, heat dumps, finding memory leak suspects

One of many examples of sharing a work-in-progress version of a chapter. Source: X

Still, the biggest benefit of sharing drafts was that I got nearly as much feedback from social media users as from beta readers.

Procrastination is real - deal with it

Writing "the book" I'd always wanted made me freeze at times: it was something I'd never done before, it was a large project, and I wanted to get everything right.

In the end, as with software, "done is better than perfect". Just like when working on a big software launch, the best way to make progress is to forget about how many people might use the product, and to just focus on the next task to complete... then the next... then the next.

It was amusing to remind myself that tech products I'd worked on were used by many times more people than the book could ever hope to reach - so why procrastinate? Plus, mistakes can be fixed with ebooks, they are trivial to do with a re-publish. Of course, this isn't possible in the same way with print books, but print-on-demand means issues can be resolved in future copies.

3. Traveling to Mongolia to meet the startup which translated my book

An unexpected highlight of publishing the book was ending up in Mongolia in June of this year, at a small-but-mighty startup called Nasha Tech. This was because the startup translated my book into the Mongolian language. Here's what happened:

A little over a year ago, a small startup from Mongolia reached out, asking if they could translate the book. I was skeptical it would happen because the unit economics appeared pretty unfavorable. Mongolia's population is 3.5 million; much smaller than other countries where professional publishers had offered to do a translation (Taiwan: 23M, South Korea: 51M, Germany: 84M people).

But I agreed to the initiative, and expected to hear nothing back. To my surprise, nine months later the translation was ready, and the startup printed 500 copies on the first run. They invited me to a book signing in the capital city of Ulaanbaatar, and soon I was on my way to meet the team, and to understand why a small tech company translated my book!

Japanese startup vibes in Mongolia

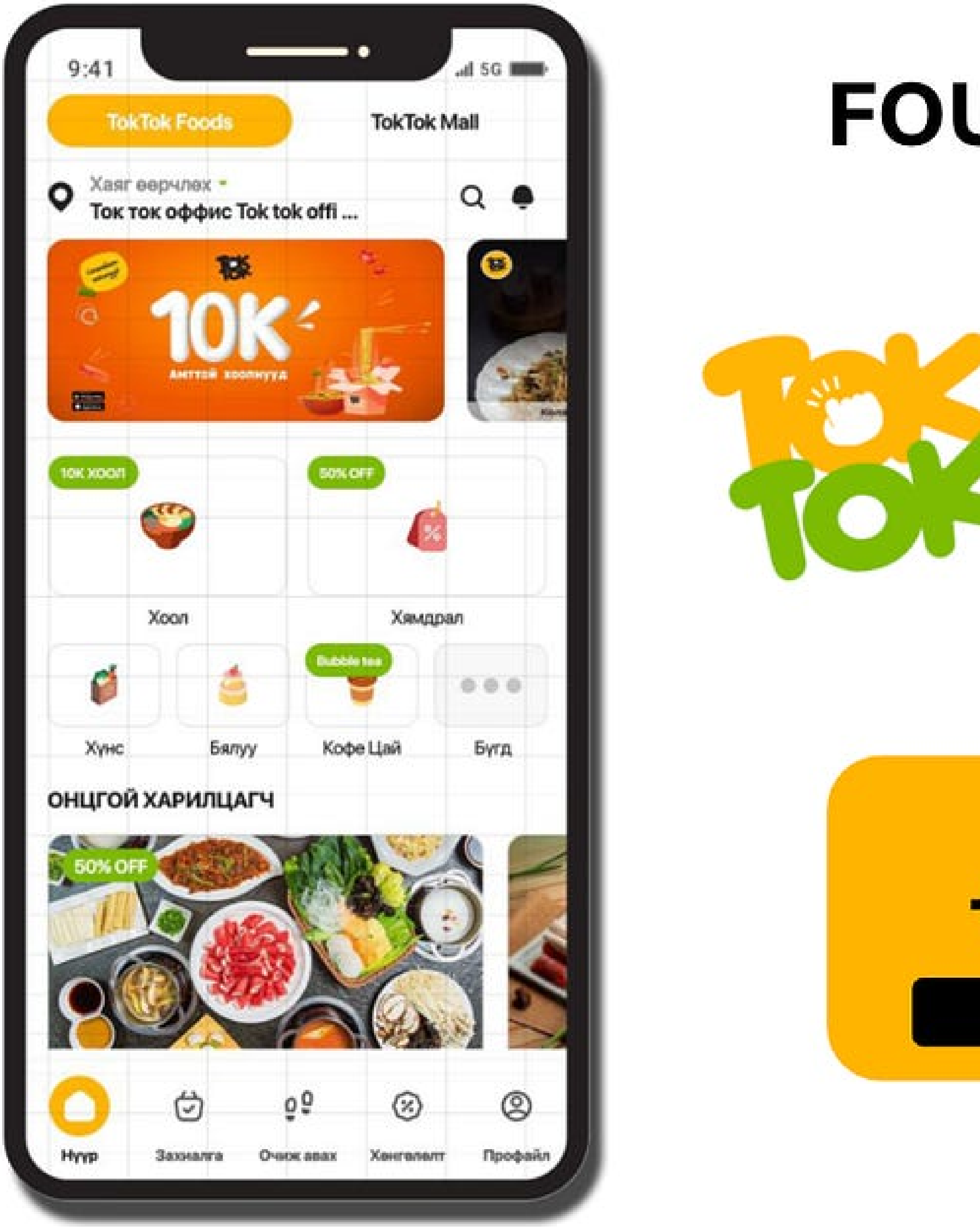
The startup behind the translation is called Nasha Tech; a mix of a startup and a digital agency. Founded in 2018, its main business has been agency work, mainly for companies in Japan. They are a group of 30 people, mostly software engineers.

Nasha Tech's offices in Ulaanbaatar, Mongolia

Their offices resembled a mansion more than a typical workplace, and everyone takes their shoes off when arriving at work and switches to "office slippers". I encountered the same vibe later at Cursor's headquarters in San Francisco, in the US.

Nasha Tech found a niche of working for Japanese companies thanks to one of its cofounders studying in Japan, and building up connections while there. Interestingly, another cofounder later moved to Silicon Valley, and advises the company from afar.

The business builds the "Uber Eats of Mongolia". Outside of working as an agency, Nasha Tech builds its own products. The most notable is called TokTok, the "UberEats of Mongolia", which is the leading food delivery app in the capital city. The only difference between TokTok and other food delivery apps is scale: the local market is smaller than in some other cities. At a few thousand orders per day, it might not be worthwhile for an international player like Uber or Deliveroo to enter the market.



The [TokTok](#) app: a customer base of 800K, 500 restaurants, and 400 delivery riders

The tech stack Nasha Tech typically uses:

Frontend: React / Next, Vue / Nuxt, TypeScript, Electron, Tailwind, Element UI

Backend and API: NodeJS (Express, Hono, Deno, NestJS), Python (FastAPI, Flask), Ruby on Rails, PHP (Laravel), GraphQL, Socket, Recoil

Mobile: Flutter, React Native, Fastlane

Infra: AWS, GCP, Docker, Kubernetes, Terraform

AI & ML: GCP Vertex, AWS Bedrock, Elasticsearch, LangChain, Langfuse

AI tools are very much widespread, and today the team uses Cursor, GitHub Copilot, Claude Code, OpenAI Codex, and Junie by Jetbrains.

I detected very few differences between Nasha Tech and other "typical" startups I've visited, in terms of the vibe and tech stack. Devs working on TokTok were very passionate about how to improve the app and reduce the tech debt accumulated by prioritizing the launch. A difference for me was the language and target market: the main language in the office is, obviously, Mongolian, and the products they build like TokTok also target the Mongolian market, or the Japanese one when working with clients.

One thing I learned was that awareness about the latest tools has no borders: back in June, a dev at Nasha Tech was already telling me that Claude Code was their daily driver, even though the tool had been released for barely a month at that point!

Why translate the book into Mongolian?

Nasha Tech was the only non-book publisher to express interest in translating the book. But why did they do it?

I was told the idea came from software engineer Suuribaatar Sainjargal, who bought and enjoyed the English-language version. He suggested translating the book so that everyone at the company could read it, not only those fluent in English.

Nasha Tech actually had some in-house experience of translation. A year earlier, in 2024, the company translated Matt Mochary's The Great CEO Within as a way to uplevel their leadership team, and to help the broader Mongolian tech ecosystem.

Also, the company's General Manager, Batutsengel Davaa, happened to have been involved in translating more than 10 books in a previous role. He took the lead in organizing this work, and here's how the timelines played out:

Professional translator: 3 months

Technical editor revising the draft translation: 1 month

Technical editing #2 by a Support Engineer in Japan: 2 months

Technical revision: 15 engineers at Nasha Tech revised the book, with a "divide and conquer" approach: 2 months

Final edit and print: 1 month

This was a real team effort. Somehow, this startup managed to produce a high-quality translation in around the same time as it took professional book publishers in my part of the world to do the same!

A secondary goal that Nasha Tech had was to advance the tech ecosystem in Mongolia. There's understandably high demand for books in the mother tongue; I observed a number of book stands selling these books, and book fairs are also popular. The translation of my book has been selling well, where you can buy the book for 70,000 MNTs (~$19).

Book signing and the Mongolian startup scene

The book launch event was at Mongolia's startup hub, called IT Park, which offers space for startups to operate in. I met a few working in the AI and fintech spaces - and even one startup producing comics.

Book launch event, and meeting startups inside Mongolia's IT Park

I had the impression that the government and private sector are investing heavily in startups, and want to help more companies to become breakout success stories:

IT Park report: the country's tech sector is growing ~20%, year-on-year. The *combined* valuation of all startups in Mongolia is at $130M, today. *It's worth remembering that location is important for startups: being in hubs like the US, UK, and India confers advantages that can be reflected in valuations.*

Mongolian Startup Ecosystem Report 2023: the average pre-seed valuation of a startup in Mongolia is $170K, seed valuation at $330K, and Series A valuation at $870K. The numbers reflect market size; for savvy investors, this could also be an opportunity to invest early. I met a Staff Software Engineer at the book signing event who is working in Silicon Valley at Google, and invests and advises in startups in Mongolia.

Mongolian startup ecosystem Map: better-known startups in the country.

Two promising startups from Mongolia: Chimege (an AI+voice startup) AND Global (fintech). Thanks very much to the Nasha Tech team for translating the book - keep up the great work!

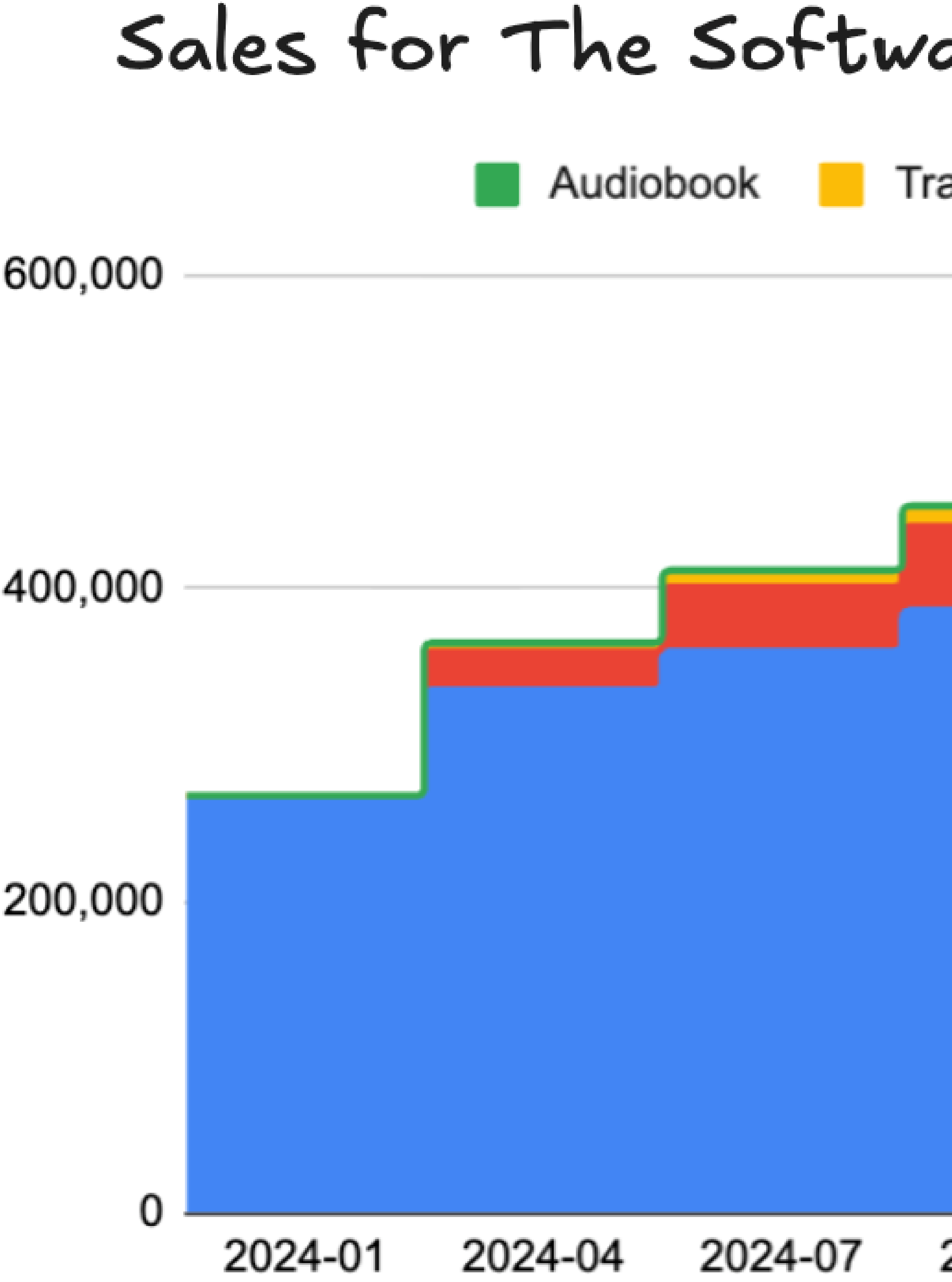
4. How much did my book earn?

There's usually little information about the key topic of how much authors make from their books being published, beyond "not much". Author Justin Garrison shared that his co-authored title Cloud Native Infrastructure earned $11,554 in its first year - and without three unexpected sponsorships, that amount would've been $3.500. Conventional wisdom states you should not write a book for money, but for the other benefits like building your status as an expert in a domain, or exploring a topic in more depth.

A notable exception to this rule is Designing Data Intensive Applications, whose author Martin Kleppman made $477,916 in royalties in the first 6 years of publication, as he shared. Martin published with O'Reilly, and the book sold 108,000 copies, generating around $4.50 per copy in royalties for the author. Still, Designing Data Intensive Applications is one of the most successful books, and Martin argues that a book's real value lies in the value it creates:

> "Writing a book is an activity that creates more value than it captures. What I mean with this is that the benefits that readers get from it are greater than the price they paid for the book. To back this up, let's try roughly estimating the value created by my book.
>
> It's hard to quantify that, but let's say that the people who applied ideas from the book avoided a bad decision that would have taken them one month of engineering time to rectify. (I'd actually love to claim that the time saving is much higher, but let's be conservative in our estimates.) Thus, the 10,000 readers who applied the knowledge freed up an estimated 10,000 months, or 833 years, of engineering time to spend on things that are more useful than digging yourself out of a mess.
>
> If I spend 2.5 years writing a book, and it saves other people 833 years of time in aggregate, that is over 300x leverage. If we assume an average engineering salary on the order of $100k, **that's $80m of value created**. Readers have spent approximately $4m buying those 100,000 books, so the value created is about 20 times greater than the value captured. And this is based on some very conservative estimates."

The Software Engineer's Guidebook will probably be another exception; it has sold 40,000 copies and netted **$611,911** in royalties in the two years since publishing:


Sales for The Softwa
🟩 Audiobook
🟨 Tra
600,000
400,000
200,000
0
2024-01
2024-04
2024-07
2

Cumulative royalties for The Software Engineer's Guidebook

Here is how revenue from different platforms adds up:

Print:

Amazon KDP: 29,806 copies, $470,000 in royalties ($16 per book)

Ingram Spark: 1,316, copies, $10,322 in royalties ($8 per book)

Ebooks:

Kindle: 3,909 copies, $42,992 royalties ($11 per book)

DRM-free ebooks: 2,713 sales, $54,963 royalties ($20 per book).

Other ebooks: 388 copies, $6,882 in royalties ($17 per book). 90% of sales came from iTunes, Google Play and Kobo.

Audiobook:

Amazon, Spotify, and other platforms: unclear how many purchases, $6,511 royalties (Audible typically pays around $1.50-3.50 per audiobook, Spotify closer to $9)

DRM-free ebook: 241 sales, $3,241 royalties ($13 per book)

I previously shared more about [how I created the audiobook]()

Translations:

5 languages, $17,000 in upfront royalty payments (South Korea: $5,000, Japan: $4,500, Germany: ?3,000, Taiwan: $2,000, China: $2,000)

Total: $611,911 in royalties, circa 40,000 in copies sold (38,373 copies, plus audiobooks, plus foreign translations that don't have exact numbers)

These are good numbers for a tech book, and it enjoyed the advantage of being published after The Pragmatic Engineer newsletter found an audience online. *Thank you to everyone who purchased a copy of the book or gifted one!*

It's interesting, as someone who self-published print and audiobook versions of their title, that royalties per book sold are highest in paperback, and much lower for the ebook and the audiobook. This is because Amazon takes 70% of the purchase price of the Kindle ebook, and 75% for the audiobook, but only 40% for the print book, in marketplace fees.

Self-publishing definitely helped substantially increase the royalties generated from the book, which would be 4-8x lower if done via a publisher. At the same time, self publishing increased the amount of time and effort I needed to invest.

5. Learnings from writing my book

The impact of a book is hard to know with certainty. When I publish a newsletter article or a podcast episode, the feedback is almost immediate: I get comments, emails, and mentions about the contents for a few days - perhaps a week or two. After that, I rarely hear feedback again.

But I run into people at events and conferences who say the book helped them focus more on their career, or get a desired promotion to senior-or-above.

A lot more readers than expected find the book via recommendations or gifting. I hear stories about my book being recommended or purchased for engineers by managers, or peers, or friends - and *then* they felt obliged to start to read it. This kind of dynamic also exists with articles and podcast episodes, but my sense is that being given a book is a *very* strong nudge to invest time in the resource.

Good books stay valuable for longer - that's why they're hard to write. The final 18 months of writing the book mostly involved editing the existing draft. I tried to remove details that were likely to age poorly in the near to medium term future - like mentions of specific frameworks, or things that felt like short-lived fads (e.g: "web3" engineering as a category, which seems to have mostly vanished from the discourse.

Even so, I got burnt by the fast-changing nature of tech. In my last edits, I added short sections on AI, about how it's a useful tool to learn languages with, or for getting unstuck. I gave examples of tools that were popular in 2023, and predicted they would remain relevant: ChatGPT, GitHub Copilot, and Google Bard. I figured that OpenAI would be around for at least 5 years, and that Microsoft and Google know how to name their products.

I was wrong: Google renamed Bard to Gemini four months after my book was published. In response, I removed all product mentions from an updated version of the book: who knows when the search giant will change the name again!

Amazon's print-on-demand service is incredibly good. Amazon prints books on demand in 10+ markets, and ships these on-demand books to even more countries. Books printed this offer the highest price-per-value across all print-on-demand services I found. Amazon's KDP is considerably more price efficient compared to the other major print-on-demand player, Ingram Spark.

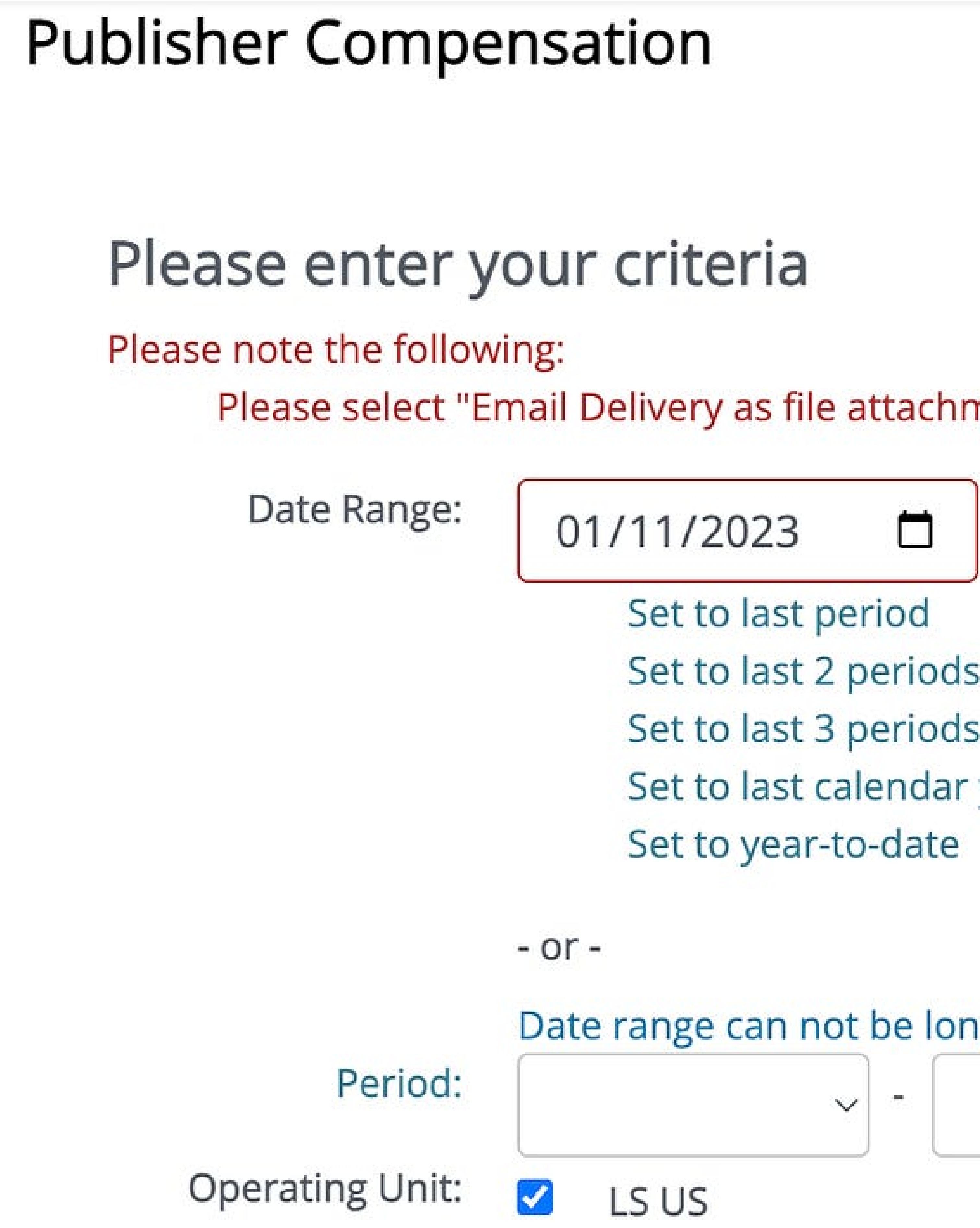
One paperback copy of my book (a 413 page book) costs $8 to print with Amazon. With Ingram Spark it's double this: $16! This is a massive difference in printing costs that is hard to ignore, and is one reason to choose Amazon's print-on-demand service as primary distribution for print books. *This explains why my royalties per book are $16 with Amazon prints, and $8 with Ingram Spark prints.*

Ebook, audiobook and print sales reporting feels stuck in the 1990s. Both ebooks and audiobooks are digital products, so you'd expect it would be possible to get near-real time sales data. But the reality is that it's not: audiobook platforms like Spotify and Audible release reports monthly (as I understand), and good luck trying to figure out which sales equate to what! This is after Audible takes 80% from sales.

Ebook reporting is similarly slow, due to sales being reported in a delayed fashion. There are so many audiobook and ebook platforms to buy on that it's only sensible to use an aggregator like Publishdrive or Voices. This adds one more layer of abstraction and more reporting delays. I understand that print sales take months to be reported, but did not expect it to be the case with audiobooks.

Even today, I have no idea how to find out how many people have listened to the audiobook, and by now I would have hoped for better reporting. It shows that if there are few suppliers in a segment (like audiobook publishers, who use these reporting tools) and not much competition: companies can get away with poor tooling.

Print reporting is perhaps even more unusable. Ingram Spark is one of the biggest print-on-demand providers - but it's not possible to get a report about a sales period longer then a year. The cherry on the cake is how if you query a period longer than 100 days, they can only email these reports:



Publisher Compensation

Please enter your criteria

Please note the following:

Please select "Email Delivery as file attachm

Date Range: 01/11/2023

Set to last period

Set to last 2 periods

Set to last 3 periods

Set to last calendar

Set to year-to-date

- or -

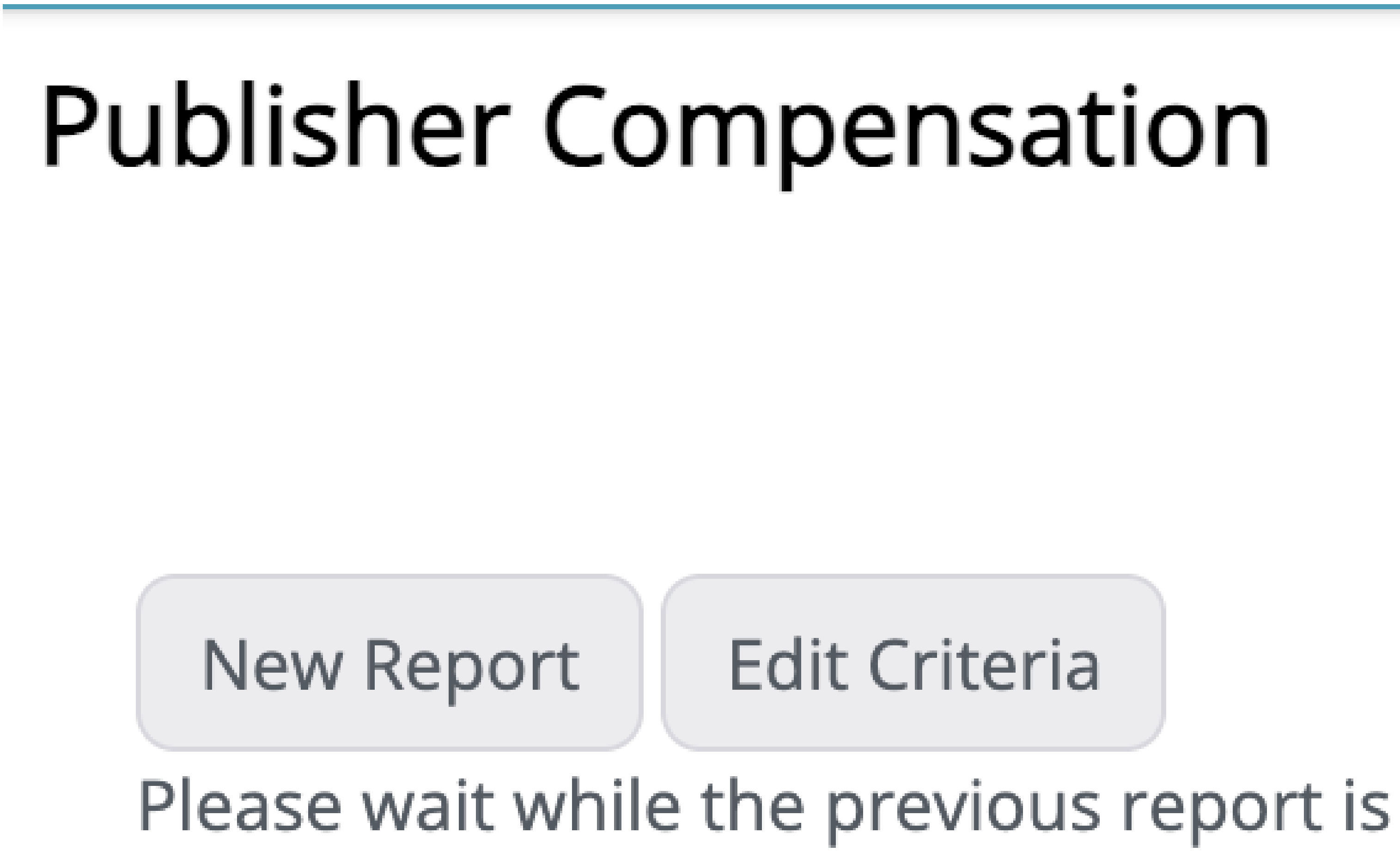
Date range can not be lon

Period: -

Operating Unit: ☑ LS US

Ingram Spark's reporting interface and functionality feels stuck 10+ years in the past. Responsive web applications, anyone?

This is a portal that has had no usability testing - and customer seem to put up with it:



Publisher Compensation

Please wait while the previous report is

Ingram Spark: instead of implementing queuing of reports, they just push the error onto the user. A hostile user experience in 2025

Amazon has an unhealthy monopoly on the audiobook and ebook sectors. Amazon generated more revenue from books sold on its site than I did, as a self-published author:

75% of revenue for all audiobooks sold via Audible

70% for all Kindle ebooks

40% for all print books as Amazon marketplace fees

That Amazon has a take rate of 75% for audiobooks and 70% for Kindle ebooks (those priced above $10) and still controls most of the market, makes this segment look like a monopoly. I offer my ebooks and audiobooks as DRM-free versions, and am happy to see more customers choose these options over the Kindle or Audible ones. Still, market forces alone don't feel strong enough to challenge Amazon's dubious pricing and practices. *I wrote more about Amazon's monopolistic audiobook practices.*

Looking back on the experience of writing my book, it was worth it for the thinking and organization that it forced on me. It has been an unusually long professional project that stretched across four years and I've learnt a lot from the process: it forced me to think deeply about topics like the importance of software architecture, and figuring out how a business works, as a Staff+ engineer. It forced me to rewrite and refine my ideas multiple times, and with each rewrite came fresh ideas and a clearer understanding of the subject. This is what really matters for software engineers to progress professionally in the industry, I believe.

Ultimately, I hope the book generates more "wealth" by helping out devs in a career rut, or who are seeking inspiration to get stronger as engineers. Obviously, commercial success is nice - and I shared those numbers in the spirit of transparency because I believe in the value of detailed, in-depth, and thoroughly researched and written information. Every anecdote helps dispel the myth that books cannot be decent earners for their authors.

If you're in the process of writing a book, why not get in touch about doing a guest post in this publication? Personally, I've found guest posts tend to be a good fit for engineers midway through writing a book!

The Pulse #152: Cursor and GitHub double down on agents

Nover

The Pulse is a series covering events, insights, and trends within Big Tech and startups. Notice an interesting event or trend? Hit reply and share it with me.

Today, we cover:

Cursor and GitHub double down on agents. Each company is focusing on agents: Cursor with its multi-agent mode, and GitHub with its "Agent HQ." Cursor is increasingly a direct rival...

From Swift to Mojo and high-performance AI Engineering with Chris Lattner

Nover

Stream the latest episode

Listen and watch now on YouTube, Spotify, and Apple. See the episode transcript at the top of this page, and timestamps for the episode at the bottom.

Jump to interesting parts:

20:28 - The story of how Swift was created

47:28 - Chris's AI learnings from working at Google and Tesla

52:24 - The Mojo programming language's origin story (Mojo is new, a high-performance, Python-compatible programming language)

1:19:00 - AI coding tools the Modular team uses (Chris's current startup)

Brought to You by

STATSIG Linear

?? **Statsig** ? - ? The unified platform for flags, analytics, experiments, and more. Companies like Graphite, Notion, and Brex rely on Statsig to measure the impact of the pace they ship. Get a 30-day enterprise trial here.

?? **Linear** - The system for modern product development. Linear is a heavy user of Swift: they just redesigned their native iOS app using their own take on Apple's Liquid Glass design language. The new app is about speed and performance - just like Linear is. Check it out.

-

In this episode

Chris Lattner is one of the most influential engineers of the past two decades. He created the LLVM compiler infrastructure and the Swift programming language - and Swift opened iOS development to a broader group of engineers. With Mojo, he's now aiming to do the same for AI, by lowering the barrier to programming AI applications.

I sat down with Chris in San Francisco, to talk language design, lessons on designing Swift and Mojo, and - of course! - compilers. *It's hard to find someone who is as enthusiastic and knowledgeable about both compilers, and programming, as Chris is!*

We also discussed why experts often resist change even when current tools slow them down, what he learned about AI and hardware from his time across both large and small engineering teams, and why compiler engineering remains one of the best ways to understand how software really works.

A quote from the episode

"I believe in the power of programmers. I believe in the human potential of people that want to create things. And that's fundamentally why I love software is that you can create anything that you can imagine."

Interesting learnings from this episode

This episode was full of interesting details. Some of my favorite ones:

"Compiles are cool. Don't let anyone tell you otherwise." My favorite quote from Chris.

Demonstrate business value inside a large company, first! Apple hired Chris to work on LLVM, because they were frustrated with GCC. However, it took Chris some friendly advice from his manager, and he "got the vibe that, after a year, if Apple's not using in some product, then they'll ask to start doing something else." This realization pushed Chris to find the first team to be the internal customer to LLVM: which was the OpenGL team, where he was able to "do something very small that actually had value." After this, more teams inside Apple onboarded to LLVM.

Chris built Swift in secret, for more than a year. For a year and a half, Chris built Swift on nights and weekends, while running a 40+ person (and growing!) team during the day.

Apple leadership was initially not enthusiastic about Swift, at all, initially. When Chris showed off an early version of Swift to leadership, the response was "why would we want a new language? Objective C is what made the iPhone successful."

A new programming language for LLMs doesn't make sense - according to Chris**.** The readability of a programming language is more important than how easy it is to write it. So in a world with more AI agents writing code, the most powerful languages will be ones that are expressive, and readable. This might be one reason Python remains so popular, and it's also how Mojo is being designed.

Why Mojo is both so readable, but also so performant: Mojo took a lot of inspiration from Python for readability, but also added features to make the language be more performant: for example, compressed floating point formats (the ability to reduce the precision of floating point numbers, by using types like Float16, and using less memory space and do calculations faster with it), and metaprogramming capabilities like parameters for compile-time calculations.

How Modular hires new grads: they look for folks who have intellectual curiosity, are fearless in getting things done, and it's a positive sign if they've contributed to open source projects before.

Chris' success is a lot more work and not listening to others than many would imagine. LLMV, Clang, Swift and other projects Chris created are used industry-wide: but early on, these projects got little support. As Chris put it: "I don't expect people to understand me. What usually happens is that these projects over time, they grow and they get to the point where suddenly it clicks and suddenly people start to understand."

What Chris is the most proud of: helping others create software. Something Chris still remembers is how, after Swift became widespread, people stopped him on the street, thanking him that they could become iOS developers using this easier to onboard language.

The Pragmatic Engineer deepdives relevant for this episode

? AI Engineering in the real world

? The AI Engineering stack

? Uber's crazy YOLO app rewrite, from the front seat

? Python, Go, Rust, TypeScript and AI with Armin Ronacher

? Microsoft's developer tools roots

Timestamps

(00:00) Intro

(02:35) Compilers in the early 2000s

(04:48) Why Chris built LLVM

(08:24) GCC vs. LLVM

(09:47) LLVM at Apple

(19:25) How Chris got support to go open source at Apple

(20:28) The story of Swift

(24:32) The process for designing a language

(31:00) Learnings from launching Swift

(35:48) Swift Playgrounds: making coding accessible

(40:23) What Swift solved and the technical debt it created

(47:28) AI learnings from Google and Tesla

(51:23) SiFive: learning about hardware engineering

(52:24) Mojo's origin story

(57:15) Modular's bet on a two-level stack

(1:01:49) Compiler shortcomings

(1:09:11) Getting started with Mojo

(1:15:44) How big is Modular, as a company?

(1:19:00) AI coding tools the Modular team uses

(1:22:59) What kind of software engineers Modular hires

(1:25:22) A programming language for LLMs? No thanks

(1:29:06) Why you should study and understand compilers

References

Where to find Chris Lattner:

? X: https://x.com/clattner_llvm

? LinkedIn: https://www.linkedin.com/in/chris-lattner-5664498a

? Website: https://nondot.org/sabre

Mentions during the episode:

? LLVM: https://llvm.org

? Swift: https://www.swift.org

? GCC: https://gcc.gnu.org

? Linux: https://www.linux.org

? Autoconf: https://en.wikipedia.org/wiki/Autoconf

? Python: https://www.python.org

? Mojo: https://www.modular.com/mojo

? REPL and Debugger: https://www.swift.org/documentation/lldb/

? Code Complete with Steve McConnell: https://newsletter.pragmaticengineer.com/p/code-complete-with-steve-mcconnell

? *Python: The Documentary*: https://lwn.net/Articles/1035537/

? CUDA: https://developer.nvidia.com/cuda-toolkit

? GPU puzzles: https://puzzles.modular.com/introduction.html

? Claude Code: https://www.claude.com/product/claude-code

? Cursor: https://cursor.com

? Beyond Vibe Coding with Addy Osmani: https://newsletter.pragmaticengineer.com/p/beyond-vibe-coding-with-addy-osmani

? *Beyond Vibe Coding: From Coder to AI-Era Developer*: https://www.amazon.com/Beyond-Vibe-Coding-AI-Era-Developer/dp/B0F6S5425Y

? Kaleidoscope Tutorial: https://llvm.org/docs/tutorial/LangImpl01.html

? Rust Compiler Development Guide: https://rustc-dev-guide.rust-lang.org/overview.html#overview-of-the-compiler

-

Production and marketing by Pen Name.