Substack Archive

Table of Contents

2025-10-08 - Python, Go, Rust, TypeScript and AI with Armin Ronacher

2025-10-09 - The Pulse #149: New trend: programming by kicking off parallel AI agents

2025-10-14 - The Pragmatic Engineer 2025 Survey: What's in your tech stack? Part 3

2025-10-15 - Google's engineering culture

2025-10-21 - What is good software architecture?

2025-10-23 - The Pulse: AWS takes down a good part of the internet

2025-10-28 - San Francisco is back as the world's leading tech hub

2025-10-29 - Beyond Vibe Coding with Addy Osmani

2025-10-30 - The Pulse #151: Amazon layoffs - AI or economy to blame?

2025-11-04 - Tech jobs market 2025, part 3: job seekers' stories

2025-11-05 - From Swift to Mojo and high-performance AI Engineering with Chris Lattner

2025-11-06 - The Pulse #152: Cursor and GitHub double down on agents

2025-11-11 - The Software Engineer's Guidebook: a recap

2025-11-12 - Netflix's Engineering Culture

2025-11-13 - The Pulse #153: Is Microsoft too early to agentic OS - like with smartphones?

2025-11-18 - Career paths for software engineers at large tech companies

2025-11-19 - How AI will change software engineering - with Martin Fowler

2025-11-20 - The Pulse #154: Cloudflare takes down half the internet - but shares a great postmortem

2025-11-25 - Holiday gift ideas for techies

2025-11-26 - Code security for software engineers

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

Octob

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. Most teams end up in this situation: ship a feature to 10% of users, wait a week, check three different tools, try to correlate the data, and you're still unsure if it worked. The problem is that each tool has its own user identification and segmentation logic. Statsig solved this problem by building everything within a unified platform. Check out Statsig.

?? **Linear** - The system for modern product development. In the episode, Armin talks about how he uses an army of "AI interns" at his startup. With Linear, you can easily do the same: Linear's Cursor integration lets you add Cursor as an agent to your workspace. This agent then works alongside you and your team to make code changes or answer questions. You've got to try it out: give Linear a spin and see how it integrates with Cursor.

-

In this episode

For an AI startup today, what language would be the best choice: Python, TypeScript, Go or Rust?

Armin Ronacher is the creator of the **Flask** framework for **Python**, was one of the first engineers hired at **Sentry**, and now the co-founder of a new startup. He has spent his career thinking deeply about how tools shape the way we build software.

In this episode of *The Pragmatic Engineer Podcast*, he joins me to talk about how programming languages compare, why Rust may not be ideal for early-stage startups, and how AI tools are transforming the way engineers work. Armin shares his view on what continues to make certain languages worth learning, and how agentic coding is driving people to work more, sometimes to their own detriment.

We also discuss:

Why the Python 2 to 3 migration was more challenging than expected

How Python, Go, Rust, and TypeScript stack up for different kinds of work

How AI tools are changing the need for unified codebases

What Armin learned about error handling from his time at Sentry

And much more

Jump to interesting parts:

06:53. - How Python, Go, and Rust stack up and when to use each one

30:08 - Why Armin has changed his mind about AI tools

50:32 - How important are language choices from an error-handling perspective?

Interesting quotes from the episode

On how Armin uses AI tools (at 28:05)

"Over the years, many of my projects didn't go anywhere because the legwork needed to build this bespoke tooling to be able to pull this project off was too much.

One of the big changes I'm noticing for myself is that I have so many better tools around now. For example, I moved from Terraform to Pulumi for one of the services just to see how that goes. I had Claude build me a perfect

control system to get my logs and visualize what's going on in production. I would never have done this before [Claude Code] just because it wouldn't have worked."

Why Armin wants to hire more engineers at his startup, despite using an "army of AI interns" until now (at 33:48:)

Now there's an argument to be made that maybe systems like Claude will help you, and maybe you can even bootstrap a startup around it. But if you fully delegate everything that you're doing to a machine, then the person who doesn't do that has an edge on you.

This is one of the reasons I want to hire more people now. It's just because Claude is not a human. There's energy to a well-running company and to motivated people that just a computer can't replace. So a big part that sticks with you as an engineer is understanding how to create your systems, and understanding when the code it produces is more right versus more wrong. I don't think much of this has changed [since AI].

Why Armin doesn't believe strongly typed languages like TypeScript meaningfully reduce errors - based on his experience observing this at Sentry (at 52:42:)

There was a large-scale realization in JavaScript ecosystem that type checkers could get rid of a whole class of errors, because at the very least you have to explicitly check if this thing is nullable or not.

I never got the feeling that the adoption of TypeScript dramatically changed anything about JavaScript error rates. None of that adoption had any meaningful impact. If there was an impact, it's immeasurable.

I remember that the adoption of more and more complex things in the React ecosystem has just dramatically increased the types of errors. For example, hydration errors were not a thing for many years. And then all of sudden, there's a whole class of new errors coming in because of this [because of the wider adoption of React.]

The Pragmatic Engineer deepdives relevant for this episode

How Claude Code is built

Real-world engineering challenges: building Cursor

AI Engineering in the real world

The AI engineering stack

Thriving as a founding engineer: lessons from the trenches

Timestamps

(00:00) Intro

(01:34) Why the Python 2 to 3 migration created so many challenges

(06:53) How Python, Go, and Rust stack up and when to use each one

(08:35) The friction points that make Rust a bad fit for startups

(12:28) How Armin thinks about choosing a language for building a startup

(22:33) How AI is impacting the need for unified code bases

(24:19) The use cases where AI coding tools excel

(30:08) Why Armin has changed his mind about AI tools

(38:04) Why different programming languages still matter but may not in an AI-driven future

(42:13) Why agentic coding is driving people to work more and why that's not always good

(47:41) Armin's error-handling takeaways from working at Sentry

(50:32) How important is language choice from an error-handling perspective

(56:02) Why the current SDLC still doesn't prioritize error handling

(1:04:18) The challenges language designers face

(1:05:40) What Armin learned from working in startups and who thrives in that environment

(1:11:39) Rapid fire round

References

Where to find Armin Ronacher:

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

? LinkedIn: https://www.linkedin.com/in/arminronacher/

? Website: https://mitsuhiko.at/

? Blog: https://lucumr.poco

Mentions during the episode:

? Flask: https://flask.palletsprojects.com/en/stable/

? Sentry: https://sentry.io/

? Python: The Documentary:

? Rust: https://www.rust-lang.org/

? Go: https://go.dev/

? WebAssembly: https://webassembly.org/

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

? TypeScript: https://www.typescriptlang.org/

? React: https://react.dev/

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

? CodeX: https://openai.com/codex/

? 996: https://lucumr.pocoo.org/2025/9/4/996/

? Peter Steinberger on LinkedIn: https://www.linkedin.com/in/steipete/

? Ron Pressler: https://inside.java/u/RonPressler/

? Debian: https://www.debian.org/

? Red Hat: https://www.redhat.com/

-

Production and marketing by Pen Name.

The Pulse #149: New trend: programming by kicking off parallel AI agents

Octob

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:

New trend: programming by kicking off parallel AI agents. More devs are experimenting with kicking off coding agents in parallel

ACP protocol. A new protocol built by the Zed team, which tries to make it easier to build AI tooling for IDEs than the MCP protocol allows

AI security tooling works surprisingly well? AI-powered security tools seem good at identifying security flaws in mature open source projects

Is AI the only engine of US economic growth? Forty percent of US GDP this year is based on AI-related spend, while 60% of venture capital goes into AI. Hopefully, it won't end up as a bubble which bursts like in 2001

Comparing interviews at 8 large tech companies. Puneet Patwari applied to 8 major tech companies, and received 6 offers. He compares his interview experiences at Meta, Amazon, Uber, and 5 other workplaces

1. New trend: programming by kicking off parallel AI agents

Read more

The Pragmatic Engineer 2025 Survey: What's in your tech stack? Part 3

Octob

Hi - this is Gergely with the monthly, free issue of the Pragmatic Engineer Newsletter. In every issue, I cover challenges at Big Tech and startups through the lens of senior engineers and engineering leaders. If you've been forwarded this email, you can subscribe here.

Many subscribers expense this newsletter to their learning and development budget. If you have such a budget, here's an email you could send to your manager.

During April and May, we asked readers which tools you use in your stack, and your opinions of them. Previous articles in this mini-series about the results of that survey have focused on AI-related tools, IDEs, and CI/CD in Part 1, while Part 2 covered project management, collaboration tools, databases, and backend-related tools.

In the third and final part of this dive into the feedback, we cover:

Observability, monitoring, and logging. Grafana, Datadog, and Sentry all stand out, with a long list of open source tools and vendors offering observability platforms.

Oncall tooling and incident management. PagerDuty and OpsGenie remain the most-used solutions, but young startups like incident.io, Rootly, and FireHydrant are challenging them.

Feature flags, analytics, and experimentation. LaunchDarkly is the most-used solution, with a long tail of analytics, experimentation, and feature flag vendors.

Frontend and mobile tooling. React is the most popular frontend framework, and Next.js is the most popular "meta frontend framework". React Native is the most-mentioned cross-platform mobile framework. In these categories, it's not even a contest.

Developer tools. Postman is popular for API development, Sonar for static analysis, and Backstage for developer portals.

Custom-built tools. Feature flagging, A/B testing, and CI/CD systems are the most common reasons to build custom tools.

Niche tools readers love. Lesser mentioned tools that are highly rated by readers, like Tuple (remote pair programming), Tmux (terminal multiplexer), Ansible (infrastructure automation), and many others.

As mentioned, check out more on the latest survey results in other articles, including:

Part 1:

Demographics

AI tools

Most used, most-loved programming languages

Most hated (and loved) tools

IDEs and terminals

Version control and CI/CD

Cloud providers, IaaS, and PaaS

Part 2:

Most-mentioned tools

Project management

Communication and collaboration

Databases and data stores

Backend infrastructure

Load balancers

Forks of popular open-source projects

1. Observability, monitoring, and logging

The leading observability vendor in this space, Datadog, was founded in 2010 with a focus on server and infrastructure monitoring: collecting metrics from servers, cloud instances, and with AWS as a focal point. Since then, Datadog has continued adding new products, and now supports dozens of features from APM (Application Performance Monitoring - distributed tracing, service maps), through RUM (Real User Monitoring - frontend performance, user sessions), to CI/CD visibility (pipeline monitoring, test analytics), and dozens more products. Datadog lists more than 50 on its website, and most are integrated to work together.

Other platforms are similar: they started with a narrow focus, added more capabilities, and became the full-blown platforms of today. A good example is Sentry: it started in 2008 as an open source monitoring tool for the Django platform in Python, then got incorporated in 2015 as a company, and today has products for APM (Application Performance Monitoring), error monitoring, session replay, and more.

With the caveat that comparing vendors in this crowded space is no simple task, below are listed the most-mentioned observability platforms by 3,000 respondents. (Thanks again to everyone who filled in the survey!):

Observa

Grafana
Datadog
Sentry
Prometheus
ELK Stack
New Relic
Splunk
AWS CloudWatch
OpenTelemetry
Loki
Honeycomb
Dynatrace
Tempo
Jaeger
Graylog
Coralogix

Observability platforms, ranked by readers

Open source tools: several of these tools are open source (meaning they have permissive licenses):

Grafana: an open source tool for displaying dashboards for observability. It's often used with Prometheus, Elasticsearch, and other backend observability databases. *Grafana Labs (creator of the project) offers a managed service and additional products like Loki and Tempo.*

Prometheus: an open-source metrics monitoring system originally built at SoundCloud. It's the de facto standard for Kubernetes monitoring, and often paired with Grafana for visualization.

The ELK stack: this stands for "Elasticsearch, Logstash, Kibana", also known as the "Elastic stack". These tools are designed to work with one another. Elasticsearch and Kibana moved from being open source in 2021 to a more restrictive license, before returning to open source in 2024. Logstash has always been open source. *In the survey, we saw 102 mentions of Kibana, 90 for Elasticsearch, 78 for the ELK stack, and 16 for Logtash. We assumed all mentions refer to the ELK stack. In an earlier version of this article, we separated these: this has now been updated.*

OpenTelemetry: an open-source observability framework providing standardized APIs, SDKs, and tools for instrumenting applications to collect traces, metrics, and logs.

Elasticsearch is an open-source distributed search and analytics engine, designed for storing and querying large volumes of data in near-real time.

Loki: an open-source log aggregation system created by Grafana Labs, inspired by Prometheus and designed for logs, instead of metrics. Integrates nicely with Grafana for visualization and with Prometheus for unified observability.

Tempo: an open-source distributed tracing backend that's designed for massive scale and cost-effectiveness by Grafana Labs. It integrates nicely with Grafana and Prometheus.

Jaeger: an open-source distributed tracing system originally developed by Uber for monitoring microservices-based architectures. Commonly used as a backend for OpenTelemetry tracing data.

Zabbix: an open-source enterprise monitoring solution for networks, servers, and infrastructure components that's popular in more traditional environments.

SigNoz: an open-source, full-stack observability platform that provides APM (Application Performance Monitoring), distributed tracing, metrics, and logs in a single application. Built as an open-source alternative to commercial platforms like Datadog or New Relic, using OpenTelemetry for instrumentation.

Grafana coupled with Prometheus is a popular combination for companies to build an in-house observability stack. This is what Shopify did when it moved off Datadog - and it was also the stack which Coinbase built after receiving an eye-watering $65M bill from Datadog in 2023.

Some vendors and solutions in the observability platforms table above are not permissive open source (several of them have source-available licenses):

Datadog: the market-leading observability vendor.

Sentry: an error tracking and application monitoring platform (APM).

New Relic: a full-stack observability platform. In 2023, New Relic published a blog post saying customers should consider leaving vendors acquired by private equity (PE). As fate would have it, 6 months later it was acquired by PE.

Splunk: another fullstack observability platform.

AWS CloudWatch: AWS's native monitoring and observability service.

Kibana: an open source tool for dashboards, charts, and maps. Often used with an Elasticsearch backend.

Honeycomb: an observability platform designed for debugging complex distributed systems with high-cardinality analysis and exploratory queries.

Dynatrace: a software intelligence platform providing automatic observability across applications and infrastructure.

Graylog: a source-available log management platform, positioned as a more user-friendly, cost-effective alternative to

Splunk and ELK Stack.

Logstash: a source-available data processing pipeline. It's the "L" in ELK Stack, and handles the collection and enrichment layer of log management.

Coralogix: observability platform that focuses on real-time analytics and cost optimization.

Papertrail: a cloud-based log management service. Popular with smaller teams and startups looking for simple log management.

Sumo Logic: a SaaS platform for log management, observability, and security.

Chronosphere: an observability platform purpose-built for microservices and containerized environments, with a focus on cost efficiency. We previously covered how Chronosphere is built, with cofounder and CEO Martin Mao.

Airbrake: an error monitoring and application performance tracking tool (APM).

2. Oncall tooling and incident management

Oncall tooling an

Incident management-related tools mentioned by readers

Tools which readers use:

PagerDuty: market leader in incident response and on-call management.

Opsgenie: Atlassian's on-call management and alerting. In 2022, OpsGenie was fully down for 2 weeks for hundreds of customers during Atlassian's longest-ever outage.

incident.io: oncall, incident response, and status pages *(note: I'm an investor).*

Grafana IRM: incident response and management built into the Grafana stack

Rootly: a Slack-native incident management platform.

FireHydrant: an incident management platform.

Pingdom: an uptime monitoring and website performance tool. While primarily a monitoring tool, it's often used as part of oncall workflows for uptime tracking.

Alertmanager: the alerting component of the Prometheus ecosystem that handles alerts from Prometheus servers and routes them to appropriate receivers. Popular in Kubernetes environments as part of the Prometheus monitoring stack.

VictorOps: an incident management platform, today called 'Splunk On-Call' after Splunk acquired it.

Squadcast: an incident management platform. It positions itself as an affordable alternative to PagerDuty.

Statuspage: Atlassian's status page solution to communicate service status and incidents to customers.

A new generation of incident management tools appears to be challenging the old guard. PagerDuty was founded in 2009 and OpsGenie in 2012, and these two tools are still the most mentioned in our survey. However, a group of startups is emerging as challengers:

incident.io: founded 2021

Rootly: founded 2020

FireHydrant: founded 2018

What seems to have shaken things up and helped the "new generation" of incident management startups spread, was PagerDuty and OpsGenie being slow to add support for handling incidents inside Slack. incident.io, Rootly, and FireHydrant gained initial momentum by offering this.

Today, pretty much all the above oncall tooling vendors describe themselves on their landing pages as "AI platforms" or "AI-first operations platforms", as they're experimenting with new features built on top of LLMs like incident summaries, or taking a first stab at root cause analysis using LLMs.

Note: the above list was updated on 18 October 2025, adding Grafana IRM, which we incorrectly omitted in the original version.

3. Feature flags, analytics, and experimentation

Shipping features behind a feature flag and A/B testing features are common practices for products with many users. The tools most mentioned in this category:

Feature flags, e

Most-mentioned feature flag and experimentation products

Vendors cited:

LaunchDarkly: the clear leader in feature flag management. Supports things like progressive delivery with percentage-based rollouts, kill switches, and more.

Amplitude: a product analytics platform that helps teams understand user behavior. It's an experimentation platform that supports feature flags.

Mixpanel: a product analytics platform focused on tracking user interactions and analyzing product usage.

PostHog: an open-source product analytics platform that combines analytics, feature flags, session replay, and experimentation in one place. It can be self-hosted or used as a cloud service.

Unleash: an open-source feature flag management platform that can be self-hosted or used as a managed service.

Statsig: a feature management and experimentation platform founded by former Meta engineers who built Facebook's experimentation infrastructure. It was acquired by OpenAI for $1B, last month.

GrowthBook: an open-source feature flagging and experimentation platform.

Optimizely: an experimentation platform targeting enterprise customers.

Flagsmith: an open-source feature flag and remote config platform that can be self-hosted or used as a managed service.

Harness: a software delivery platform that includes feature flags as part of a broader continuous delivery and DevOps suite.

ConfigCat: a feature flag service focused on simplicity and transparent pricing. Popular with smaller teams and startups looking for an easy-to-use feature management tool.

Split: a feature delivery platform combining feature flags with detailed impact analysis and experimentation capabilities.

Feature flags are the most likely tool to be built in-house, as we'll see in the "Custom tools" section below.

4. Frontend and mobile tooling

The most-used frontend frameworks by respondents to our survey:

Frontend fra

Most-mentioned frontend frameworks

React continues to be the most popular by far, and not much has changed since 2022 when 76% of web developers used it, as we covered at the time in the State of Frontend 2022.

"Meta frameworks" are frameworks built on top of one of the above technologies. They add features like server-side rendering (SSR), static site generation (SSG), routing, data fetching, and other features that most production applications need, but which core frameworks don't provide.

The most popular meta frameworks in the responses:

Next.js (163 mentions): a React framework that provides server-side rendering, static site generation, API routes, and automatic code splitting out of the box. Built by Vercel.

Nuxt (29 mentions): a Vue.js framework that provides server-side rendering, static site generation, and automatic routing.

Remix (24 mentions): a full-stack React framework focused on web standards, progressive enhancement, and optimized data loading.

Astro (16 mentions): a modern static site builder that ships zero JavaScript by default, and supports using multiple frameworks in one project.

Other frontend tooling

CSS frameworks and styling:

Tailwind CSS (170 mentions): provides low-level utility classes for building custom designs without writing custom CSS. Seems to be becoming the preferred choice for many frontend teams.

Bootstrap (52 mentions): a popular CSS framework providing pre-built responsive components, utilities, and a flexible grid system based on Flexbox.

Material-UI (MUI) (44 mentions): a React component library implementing Google's Material Design specification.

Sass/SCSS (29 mentions): Sass is a CSS preprocessor that extends CSS with variables, nesting, mixins, and functions for more maintainable stylesheets. SCSS syntax is CSS-compatible, and it compiles to standard CSS.

Package managers:

npm (34 mentions): the default package manager for Node.js, which also happens to be the world's largest software registry.

pnpm (16 mentions): a fast, disk-space efficient, package manager. It saves significant disk space by sharing packages across projects. It's getting more popular among developers and companies with large monorepos and multiple projects.

yarn (13 mentions): a fast, reliable, and secure package manager developed by Meta. Popular for larger projects and monorepos.

Build tools:

Vite (75 mentions): a modern build tool with extremely fast development server startup.

Webpack (20 mentions): formerly the industry-standard bundler. It's less popular since faster build tools like Vite and Bun appeared.

Bun (14 mentions): a very fast, all-in-one, JavaScript runtime, bundler, test runner, and package manager written in the programming language of Zig. It offers dramatically faster performance than npm (for package management) and webpack (for builds).

Testing tools:

Playwright (36 mentions): a modern end-to-end testing framework by Microsoft supporting multiple browsers (Chromium, Firefox, WebKit) with a single API.

Jest (31 mentions): a JavaScript testing framework developed by Meta

Cypress (22 mentions): an end-to-end testing framework with time-travel debugging and automatic waiting.

Vitest (20 mentions): a very fast unit test framework powered by Vite. Increasingly the preferred testing framework for Vite projects and modern JavaScript applications.

CDNs: content delivery networks (CDNs) speed up web and mobile applications by caching frequently accessed resources like images and video streams. There are three major players in this field: Cloudflare, Fastly, and Akamai. This is how they compare, based on mentions by 114 respondents:

CD

Split of mentions: Cloudflare (100 mentions), Fastly (7) and Akamai (7)

iOS and Android

In the survey, there's an almost equal number of engineers working with iOS (400 mentions) as on Android (398 mentions) - a remarkably even split!

iOS vs

An almost exactly even split between devs working with iOS and those working with Android

The most popular iOS languages and frameworks, by number of mentions:

Swift: Apple's modern programming language for iOS, macOS, watchOS, and tvOS development, replacing Objective-C. Five times more developers say they use Swift (162) than who use Objective C (32)

Xcode: Apple's IDE for creating apps across all its platforms. It includes code editors, debugging tools, Interface Builder, simulators, and testing frameworks in one comprehensive package. An essential tool for iOS and macOS developers.

SwiftUI: Apple's declarative framework for building user interfaces across all its platforms with Swift code. It enables faster development with live previews, automatic dark mode support, and seamless integration with existing UIKit apps. Rapidly gaining adoption since its 2019 release, especially for new projects.

Objective C: increasingly replaced by Swift when building iOS applications.

Most-mentioned Android languages and frameworks:

Kotlin: the preferred programming language for modern Android development. Google announced it as its preferred language for Android in 2019.

Java: increasingly replaced by Kotlin for modern Android development.

Android Studio: Google's official IDE for Android development, built on IntelliJ IDEA.

Jetpack Compose: Android's modern declarative UI toolkit for building native interfaces with Kotlin.

Cross-platform frameworks

The most commonly mentioned cross-platform frameworks in our survey:

Cross-platform mo

React Native is the most popular cross-platform framework among respondents

Based on this data, React Native is the most popular choice for cross-platform development among respondents at startups, scaleups, and in Big Tech. This mirrors what we see with startups and scaleups choosing React Native more often. It's more popular than Flutter in the US and UK, as we cover in the deepdive, Cross-platform mobile development.

5. Developer tools

API development, testing, and management

API developer tools:

Postman (105 mentions): an API development platform to design, test, document, and monitor APIs.

Insomnia (8 mentions): a REST and GraphQL client to test and debug APIs.

cURL (7 mentions): a tool to make API requests from the command line

Bruno (6 mentions): an API client that is Git-integrated, fully offline, and open-source

API gateways: these handle tasks like authentication, request routing, rate limiting, and monitoring. Common ones mentioned:

AWS API Gateway (90 mentions): an AWS service that acts as a reverse proxy to accept all API calls, aggregate various services, and return the appropriate result.

Kong (48 mentions): an open-source API gateway and microservices management layer.

GCP Apigee (26 mentions): Google's full-lifecycle API platform to design, secure, deploy, monitor, and scale APIs.

Apache APISIX (5 mentions): an open source, high-performance API gateway.

Static analysis

Sonar stands out as the most-oft mentioned tool for static analysis:

Static a

Sonar & SonarQube
Synk
ESLint
Prettier
Ruff
Checkmarx
Mypy
Trivy

Static analysis tools

Tools that readers include in their responses:

Sonar/SonarQube: a code quality and security analysis platform.

Snyk: a platform that scans code, dependencies, containers, and infrastructure-as-code, for vulnerabilities.

ESLint: a linting tool for JavaScript and TypeScript to catch potential errors, and enforce coding standards.

Prettier: an opinionated code formatter that enforces a consistent coding style.

Ruff: a fast Python linter and code formatter written in Rust.

Checkmarx: static application security testing.

Mypy: a static type checker for Python.

Trivy: a security scanner for containers to detect vulnerabilities and misconfiguration.

PHPStan: static analysis tool for PHP.

RuboCop: static code analyzer and code formatter for Ruby.

Developer portals

As teams grow, developer portals start to become useful for service discovery and API directories. The most frequently mentioned tools in our survey with this functionality:

ServiceNow *(56 mentions): devs use ServiceNow for ticketing (instead of e.g., JIRA), on-call (an alternative to things like PagerDuty), and service management (rather than Backstage, for example). Survey responses are somewhat negative about ServiceNow's complexity, but it is the go-to solution in large enterprises for Information and Communication Technology (ICT) management, as it allows setting up custom workflows, and supports clear audit trails.*

Backstage (48 mentions): an open source platform by Spotify for building developer portals, and to offer a unified frontend for all infrastructure tooling and services. *We published a deepdive into Backstage.*

Custom-built internal developer portals (17 mentions): internally-built developer portals, usually made by internal platform teams for use in things like managing home-grown feature flags and deploying systems.

Compass (9 mentions): Atlassian's developer experience platform that allows tracking engineering output and health. Offers service catalogs, scorecards, and metrics to understand software quality and developer productivity.

Code review

GitHub, GitLab, and Bitbucket all have code review functionality, but readers tell us about other smaller tools which they rate for their code review capabilities:

Graphite (43 mentions): a modern code review tool that supports stacked pull requests and faster code reviews with a command-line interface, and now with AI code review as well.

CodeRabbit (13 mentions): an AI-powered code review assistant that automatically reviews pull requests and provides AI-augmented feedback.

Gerrit (12 mentions): an open source, web-based code review tool originally developed for Android, which integrates nicely with Git.

Phabricator (10 mentions): an open-source suite of web-based software development collaboration tools originally created by Meta. It has not been maintained since 2021, but a few companies still use it via forks like **Phorge**.

Graphite, Gerrit, and Phabricator all support the stacked diff workflow. We covered more on why this is a useful way to work in the deepdive, Stacked Diffs (and why you should know about them).

6. Custom-built tools

We asked readers what kind of home-grown or custom tools their team or company uses. And your responses are varied! Here's a summary of the most popular custom-built tools mentioned in the survey:

source
open
serv
tooling
f
ui
test
manageme
k8s
ci
interna
aws
build
data
s
feature flag
api
web
platform
dev
deploymen
experimentati
monitoring
database
manage

Word cloud of home-grown / custom-built tools used by readers

What stands out:

Feature flagging. According to our survey, the type of system that's most often custom-built in house is for feature flagging. I'd speculate this is because it's easy to build a first version, and vendors can be expensive. Of course, *maintaining* such systems is where things get tricky, but large-enough companies can have a platform team own it.

A/B testing and experimentation. Another common mention. Integrating A/B testing into internal systems - such as tools for the data team to analyze results, or to integrate with the feature flag system - is a common reason to build in-house.

CI/CD systems. Using custom continuous integration / continuous deployment systems is common enough, although it's more common to integrate software like Jenkins and run it on existing infrastructure, instead of building "truly" custom software.

Logging systems: another pretty common mention. As with feature flags, it's easy enough to write your own - at least in the beginning.

7. Niche tools readers love

To close, let's spotlight some lesser-mentioned tools that some respondents say they love. Why not consider giving them a spin when facing a problem they might solve?

Tuple: remote pair programming app

"Tuple is amazing! I work fully remotely and my team pair on everything, so good tooling for remote pairing is essential. The video quality is great, I never have problems with screen resolution (which I do when sharing my large monitor on Google Meet, when my colleague is viewing on a laptop screen). It's easy to point at things, annotate your pair's screen, and even take control when you need to". - Senior engineer at a mid-sized company.

Tmux: a terminal multiplexer

"Tmux makes my work extremely efficient. I want to fire up a server, open a new window or a pane, query my database, and open a new pane. I can do whatever I want without moving my eyes from the screen and my hands from the keyboard". - Senior engineer at a startup.

Ansible: infrastructure automation

"Ansible lets me 'do stuff' to multiple hosts all at once. It's great!" - Senior security engineer at a large company.

Sourcegraph: code search

"It makes it so easy and intuitive to find relevant examples or check on usage of code, especially because of our heavily distributed codebase with thousands of repos and microservices" - Software engineer at a large publicly traded company.

Argo: a Kubernetes-native workflow orchestration engine

"I love Argo Workflows, because it is a Kubernetes-native orchestration engine. This means it is lightweight and allows easy use of other Kubernetes tools without 'hacking' the solution". - Tech lead at a small company.

Beyond Compare: a file/folder difference comparison tool:

"It does its job so well and intuitively that it is one of the first tools after the code editor I install on any new PC I use". - VP of Engineering at a large company.

Obsidian: a note-taking app:

"It's so flexible and powerful: it's like coding but for notes". - Senior frontend engineer at a small business.

Dagster: data orchestration:

"It understands that data is declarative by nature. A really awesome and new approach to data orchestration". - Engineering manager at a quant fund.

Excalidraw: sketching diagrams:

"I'm personally a big fan of Excalidraw, it does one thing and does it really well". - Tech lead at a startup.

Doppler: secrets management

"A fantastic secrets manager. It gets rave reviews from our team!" - Director of engineering at a publicly-traded health &

fitness company.

Elixir: a programming language:

"It's simple, clean, productive, and doesn't break every other day with new updates". - Senior engineer at a startup.

Superhuman: an email tool:

"Sadly, I get too many emails, but Superhuman helps me a lot to tame my inbox. UX is incredible - if you like using a keyboard". Engineering lead at a startup.

uv: a blazing fast package manager for Python, written in Rust:

"uv has come out of nowhere and it has made Python version management and dependency management a breeze". - VP of engineering at a small company.

Warp: a terminal with AI integrations:

"Its built-in AI is irreplaceable for me now, lots of UX delights". - Staff Engineer at a large tech company.

Takeaways

I hope you've found this article and the two others in this mini-series about our survey results useful. What stands out for me in this final set of responses is:

Observability and logging are tough problems. Leading vendors offer a large number of products that integrate with each other, but open source products for personally building and operating an observability stack are equally popular.

Of course, there is no single ideal solution: every team needs to figure out which tools work *well enough* for them.

I sense less innovation in frontend and mobile these days. React is as popular as ever, and Angular and Vue are the stable next choices for a minority of developers. Across Android and iOS, the most popular frameworks mentioned were released in 2019 (SwiftUI for iOS, Jetpack Compose for Android).

Developer portals' momentum seems to have stalled. Backstage was mentioned by 1 in 60 respondents, suggesting most companies don't have service discoverability issues, or they solve it with internal, custom tools.

As developers, we can (and do!) build custom tools to solve problems. I really enjoyed learning from readers about their custom, internal tools built for problems like feature flagging, CI/CD, and others. Some tech leaders and executives might argue that building in-house is not a great tradeoff, and ask why not just buy solutions from vendors.

But at the end of the day, we're engineers.

Sometimes, when a good-enough solution isn't available, we are more than capable of building our own! If we always defaulted to choosing a vendor, we'd miss out on the pleasure of sharpening our skill at building from scratch.

Thanks again to everyone who filled out this year's survey about tech stacks. Add a comment below if there's a tool you want to mention, or if you have an opinion you'd like to share.

Leave a comment

Google's engineering culture

Octob

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. Something interesting is happening with the latest generation of tech giants. Rather than building advanced experimentation tools themselves, companies like Anthropic, Figma, Notion and a bunch of others... are just using Statsig. Statsig has rebuilt this entire suite of data tools that was available at maybe 10 or 15 giants until now. Check out Statsig.

?? **Linear** - The system for modern product development. Linear is just so fast to use - and it enables velocity in product workflows. Companies like Perplexity and OpenAI have already switched over, because simplicity scales. Go ahead and check out Linear and see why it feels like a breeze to use.

-

In this episode

What is it *really* like to be an engineer at Google?

In this special deep dive episode, we unpack how engineering at Google actually works. We spent months researching the engineering culture of the search giant, and talked with 20+ current and former Googlers to bring you this deepdive with **Elin Nilsson**, tech industry researcher for **The Pragmatic Engineer** and a former Google intern.

Google has always been an engineering-driven organization. We talk about its custom stack and tools, the design-doc culture, and the performance and promotion systems that define career growth. We also explore the culture that feels built for engineers: generous perks, a surprisingly light on-call setup often considered the best in the industry, and a deep focus on solving technical problems at scale.

If you are thinking about applying to Google or are curious about how the company's engineering culture has evolved, this episode takes a clear look at what it was like to work at Google in the past versus today, and who is a good fit for today's Google.

Jump to interesting parts:

(13:50) Tech stack

(1:05:08) Performance reviews (GRAD)

(2:07:03) The culture of continuously rewriting things

For more reading on this topic, see our previous deepdive articles on Google's engineering culture.

The Pragmatic Engineer deepdives relevant for this episode

Inside Google's engineering culture

Oncall at Google

Performance calibrations at tech companies

Promotions and tooling at Google

How Kubernetes is built

The man behind the Big Tech comics: Google cartoonist Manu Cornet

Timestamps

(00:00) Intro

(01:44) Stats about Google

(11:41) The shared culture across Google

(13:50) Tech stack

(34:33) Internal developer tools and monorepo

(43:17) The downsides of having so many internal tools at Google

(45:29) Perks

(55:37) Engineering roles

(1:02:32) Levels at Google

(1:05:08) Performance reviews (GRAD)

(1:13:05) Readability

(1:16:18) Promotions

(1:25:46) Design docs

(1:32:30) OKRs

(1:44:43) Googlers, Nooglers, ReGooglers

(1:57:27) Google Cloud

(2:03:49) Internal transfers

(2:07:03) Rewrites

(2:10:19) Open source

(2:14:57) Culture shift

(2:31:10) Making the most of Google, as an engineer

(2:39:25) Landing a job at Google

References

Where to find Elin Nilsson:

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

? LinkedIn: https://www.linkedin.com/in/hejelin/

? Website: https://hejelin.github.io/

Mentions during the episode:

? What I Learned Getting Acquired by Google: https://shreyans.org/google

? What is Reliability Engineering?: https://newsletter.pragmaticengineer.com/p/reliability-engineering

? Borg: https://research.google/pubs/large-scale-cluster-management-at-google-with-borg/

? Kubernetes: https://kubernetes.io/

? The Production Environment at Google, from the Viewpoint of an SRE: https://sre.google/sre-book/production-environment/

? Jeff Bezos Was An Early Investor In Google - A Move That Could Have Made Him A Billionaire Outside Of Amazon. He Took A Gamble Because He Believed In The Founders: 'I Just Fell In Love With Larry And Sergey': https://finance.yahoo.com/news/jeff-bezos-early-investor-google-185940547.html

? B4 and After: Managing Hierarchy, Partitioning, and Asymmetry for Availability and Scale in Google's Software-Defined WAN: https://research.google/pubs/b4-and-after-managing-hierarchy-partitioning-and-asymmetry-for-availability-and-scale-in-g

oogles-software-defined-wan/

? *Site Reliability Engineering*: https://sre.google/sre-book/table-of-contents/

? Piper: https://en.wikipedia.org/wiki/Piper_%28source_control_system%29

? Why Google Stores Billions of Lines of Code in a Single Repository: https://research.google/pubs/why-google-stores-billions-of-lines-of-code-in-a-single-repository/

? Code Search: https://sourcegraph.com/code-search

? Efficient scaleups in 2024 vs 2021: Sourcegraph (with CEO & Co-founder Quinn Slack): https://newsletter.pragmaticengineer.com/p/quinn-slack-efficient-scalups

? Bazel: https://opensource.google/projects/bazel

? Cider: https://code.google.com/archive/p/cider-ide/

? Tricorder: Building a Program Analysis Ecosystem: https://research.google/pubs/tricorder-building-a-program-analysis-ecosystem/

? Issue tracker: https://issuetracker.google.com/i

? GUTS: https://play.google.com/store/apps/details?id=com.vdkgroep.guts

? gRPC: a true internet-scale RPC framework is now 1.0 and ready for production deployments: https://cloud.google.com/blog/products/gcp/grpc-a-true-internet-scale-rpc-framework-is-now-1-and-ready-for-production-deployments

? What I Hated about working at Google:

? NeetCode: https://neetcode.io/

? The man behind the Big Tech comics - with Manu Cornet: https://newsletter.pragmaticengineer.com/p/the-man-behind-the-big-tech-comics

? Welcome to Google comic: https://goomics.net/50

? Oncall Compensation for Software Engineers: https://blog.pragmaticengineer.com/oncall-compensation/

? GRAD: https://www.google.com/about/careers/applications/buildyourfuture/programs/grad

? Performance Calibrations at Tech Companies: Part 1: https://newsletter.pragmaticengineer.com/p/performance-calibrations

? Performance Calibrations at Tech Companies: Part 2: https://newsletter.pragmaticengineer.com/p/performance-calibrations-part-2

? Promotions and tooling at Google (with Irina Stanescu, Ex-Google): https://newsletter.pragmaticengineer.com/p/promotions-and-tooling-at-google

? Why I Quit Google to Work for Myself: https://mtlynch.io/why-i-quit-google/

? Killed by Google: https://killedbygoogle.com/

? Side project time: https://en.wikipedia.org/wiki/Side_project_time

? *Software Engineering at Google*: https://books.google.com/books?id=V3TTDwAAQBAJ&lpg=PP1&pg=PP1#v=onepage&q&f=false

? John Doerr on LinkedIn: https://www.linkedin.com/in/john-doerr-03248211/

? *Measure What Matters: The Simple Idea that Drives 10x Growth*: https://www.amazon.com/Measure-What-Matters-Simple-Drives/dp/024134848X

? App Engine: https://cloud.google.com/appengine

? Go: https://en.wikipedia.org/wiki/Go_(programming_language)

? Angular: https://angular.dev/

? Tensorflow: https://www.tensorflow.org/

? Attention is All You Need: https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf

? Chromium: https://www.chromium.org/

? Flutter: https://flutter.dev/

? Inside Fast's Rapid Collapse: https://newsletter.pragmaticengineer.com/p/the-scoop-fast

? TikTok: https://www.tiktok.com/

? U.S. and Plaintiff States v. Google LLC and Apple Inc:
https://www.justice.gov/atr/case/us-and-plaintiff-states-v-google-llc-and-apple-inc

? State of the software engineering job market in 2025: what the data says:
https://newsletter.pragmaticengineer.com/p/state-of-the-tech-market-in-2025

-

Production and marketing by Pen Name.

What is good software architecture?

Octob

To try and answer this seemingly basic question, I turned to software engineering veteran Matthew Hawthorne. He's worked as a software engineer for more than 25 years, and is the author of the upcoming book Push to Prod or Die Trying, which shares lessons from the trenches on software, architecture, and pushing to production. It is in early release now and due to be published next year.

Matt has worked at Comcast, Twitter, and Netflix, where he stayed for 6 years during the 2010s. During his time at Netflix, every engineer made architectural decisions day-to-day and shipped features frequently, which rolled out to tens of millions of users - all without a single mention of the job title "Architect".

In this issue, Matt covers:

Architects are not the solution to architectural problems. For a long time, there were no Architects at Netflix, yet the foundations of the architecture were still sound.

Trading off today's problems for tomorrow's: migrating to AWS. Moving to the AWS cloud created plenty of problems for Netflix which took serious effort to resolve, down the line. But it also fixed existing pain points.

Good characteristics of architecture. These balance practical and aspirational concerns, unify people as well as systems - and are unrelated to good code.

Bad architecture is a lot of work that doesn't change much. It's like rearranging the furniture in a house that should be demolished and replaced by an improved layout.

Good architectural trades in Netflix projects. Making unusual tradeoffs, building tooling to reduce operational work, trading off one set of limitations for another, and upgrading systems to work better in the future.

How to improve your architecture skills. Design systems based on what could break, know your audience, focus on the right details, and make yourself valuable across several roles.

If you'd like to keep up with Matt's writing, subscribe to his newsletter, Push to Prod. You can also purchase his work-in-progress book Push to Prod or Die Trying, which is currently 40% complete.

With that, it's over to Matt:

At Netflix, we preferred building prototypes to writing formal architecture proposals, while at other companies, I've seen impractical architecture proposals fail, overly practical architectural efforts succeed - but then deliver limited impact - and other proposals gather dust due to the absence of the conversations and alignment work necessary for creating a shared plan.

I've always found the work defined by the phrase *software architecture* to be vague, and the value delivered by capital "A" *Architects* to be debatable. I speak from personal experience here after my own brief, unimpressive tenure as a "domain architect" - an experience from which I learned much, but delivered very little. Over time, if there's one thing I've learned about architecture, it is that:

Good architecture work is about purposefully trading problems you have today, for better problems tomorrow. Basically, if you're not upgrading your problems, you're just rearranging furniture.

1. Architects are not the solution to architectural problems

I worked at Netflix for 6 years and during that time, there were hundreds of things that made the company unique. One was the lack of software architects - in title - at least. In fact, we didn't have engineering levels at all; every engineer was a *Senior Software Engineer*.

I spent my first 4 years there on the Edge team, which was around 20 engineers building and operating a software layer that sat between client devices and backend services. "Edge" in this case was the edge of Netflix's server infrastructure, which carried a heavy operational burden.

It felt like the first step in every production incident was for us to prove we weren't to blame for it. This was annoying but effective; colleagues assumed that if every client request goes through our system, then we must have logs or metrics captured by our system that identify and clarify problems. We often did, and when we didn't, we worked to close the gaps. Of course, sometimes we *did* cause incidents, and on occasion, "we" was "me".

Over a few years, we solved the most pressing problems by increasing resiliency, building flexible edge routing, and enabling predictive autoscaling. If I reflect on these projects and others from my time at the video streaming giant, there wasn't architecture in the sense that I'd seen before. Those who proposed ideas also did the work, which avoided the awkward white-collar vs. blue-collar split that often arises between architects and engineers in tech workplaces. Most technical ideas originated in the "trenches", resulting in deliberate architectural choices with obvious value. Over time, we entered a new era that involved extensive conversations about creating massive new systems to solve imaginary future problems, instead of real, present ones.

A new behavioral archetype formed at the company - the "Architect". I am talking about engineers who hovered around the work, pushed new ideas, and didn't make material contributions to day-to-day struggles. They weren't on our on-call rotations, ostensibly because their managers wanted to give them more time to brainstorm new ideas, so they had ample time for conversations and intellectual enquiry, while the rest of us did grimy, operational work. Over lunch one day, a colleague and I discussed the situation. "I'm not sure what the solution is", I said. "The solution", he replied, "is to not have architects." I laughed, then realized he wasn't joking.

I thought Netflix didn't *need* architects, and considered this more broadly: does any company *really* need them? Let's examine why the role of Architect exists. I believe it's an artifact of the old-school waterfall fantasy:

Design: Architects think big thoughts and draw diagrams

Build: Engineers turn diagrams into code

Test: QA finds bugs

Operate: Ops keeps the systems running smoothly

These four phases are inescapable, and every company approaches them in its own unique way. Imagine an axis: on the left, there's a distinct role for each phase, and on the right, there's a single role performing them. Every company lives somewhere on this axis.

For example, the shift towards agile at big companies made "big design, up front" less popular, and often combined *Design* and *Build* into a single, continuous phase. And the shift towards cloud, DevOps, and the "you write it, you run it" model often blends *Build*, *Test*, and *Operate* into a single phase.

So, why did Netflix introduce architects? My perception is that this was related to scope: our management chain believed we needed bigger, bolder ideas to redefine how our systems worked, and that those tasked with achieving this needed distance from day-to-day work.

It sounds perfectly logical, but I've never seen it work. I didn't try to fix it - I switched teams and moved to the Personalization Infrastructure group, a domain I'd always been interested in. This team was newer and they were solving concrete problems. As I understand, the Edge team continued on the path of re-visualizing their platform with great success.

On a long enough timeline, success breeds specialization. You solve the hardest problems and management asks, "What's our 3-year vision?". Distinct groups emerge of "thinkers" and "builders", and many companies formalize this with Principal or Staff Engineers who function as architects in all but name. From what I've seen, it is hard for people in these roles to avoid becoming disconnected from reality: their minds are powerful, but structure and incentives make it difficult to find appropriate targets.

2. Trading off today's problems for tomorrow's: migrating to AWS

The most significant tradeoff I saw at Netflix involved a decision that predated my time there: moving the company's entire infrastructure to AWS. "So what?" you may think; lots of companies run everything in AWS.

Sure, lots of companies do so *today*, but none of significant scale did in 2010. I recall reading a few blog posts about Netflix's AWS migration before I joined, and thinking, "they're insane." But after I joined and saw the situation up close, it made a lot of sense.

What are the repercussions of running in a datacenter, on-prem, non-cloud environment?

Stable networking and hardware.

Fixed capacity constraints. Perhaps not *permanently* fixed, but capacity can be increased as fast as machines are purchased, racked, and put online.

What are the consequences of running in AWS or other cloud environments?

Virtually unlimited, elastic capacity.

Unstable networking and hardware. If "unstable" sounds harsh, let's call it "much less stable than desired."

If that was everything, then the business case for elastic capacity was strong enough to declare the AWS migration a solid trade. But from the perspective of the Edge team, this trade sent us down a path of many other problems.

For example:

Obsessing over resiliency by building libraries to implement circuit breakers, concurrency limits, and fallbacks, as well as dashboards to observe this behavior in real time.

Detecting and terminating bad AWS instances.

Refining service discovery and software load balancing for hundreds of elastic clusters.

Fine-tuning reactive autoscaling policies.

Abandoning reactive autoscaling and building a predictive autoscaling system from scratch.

These were problems we wouldn't have had if we'd stayed in the data center. But they were also exponentially better than having fixed-capacity software and constrained customer growth. This was not only a good tradeoff for the business - it also catalyzed a planet-scale improvement of our engineering capabilities.

3. Good architecture characteristics

Let's discuss a few characteristics of good and bad architecture work.

Good

is unre

unifies pe

Good architecture is unrelated to good code

Very early in my career, I sat in a meeting and watched a newly hired, experienced engineer draw manically on a whiteboard for 30 minutes. I had no idea what he was talking about, but he was highly respected by senior teammates, so I paid close attention.

A few weeks later, I had some free time and we paired up to prototype some of his ideas. His code was atrocious and he had a hard time getting anything to work. But he said something which struck my young mind as profound:

"We've got SQL queries hardcoded into the UI", he said. "We should change that so that the UI calls a service which abstracts the query behind a concept, like 'get all items for the current user'".

That's a great idea, I thought. Over the next few years, I learned a ton from him about testing, separation of concerns, hot and cold storage, compression, data formats and schemas, and more. But I never reconciled how a person so bad at writing code could have such solid ideas about system design. A larger point emerged:

Someone's ability to write high-quality code is entirely independent of their ability to create or recognize high-quality architecture.

I've worked for engineering leaders who hadn't written code in years, yet had a brilliant sense of technical fundamentals. When necessary, they grabbed the wheel and steered their teams away from expensive mistakes and towards bountiful opportunities. They didn't have great coding skills, but possessed impeccable design sense, instincts, and taste.

I've also worked with engineers who were experts in various programming minutiae, yet had no clear ability to build systems that could work together coherently and functionally. Their ability to optimally implement classes or functions did not extend beyond the source files they edited.

Architecture connects systems built with code, but is a distinct discipline from coding with limited overlap. You cannot LeetCode your way into a cohesive, opportunistic architecture that serves the business.

Why not? Let's explore the "good at architecture, bad at code" archetype. I've found these individuals are drawn to the most essential details. When designing a new system, they ask questions like:

What data can we collect to ensure things work correctly?

What is the maximum acceptable latency for clients of this system?

Do we need to compute results in real time, or can we do it offline in batch mode?

They are generally less concerned with details like class hierarchies, method names, and directory structures, which are trivial in the broader business and distributed systems contexts.

But code can't be completely irrelevant, right? Let's examine the "good at code, bad at architecture" archetype, which I think occurs when individuals haven't had a role in which things like network latency, fallbacks, or failure modes are first-class citizens.

Note, I use the word "architecture" to mean "distributed systems architecture", which is common and also biased towards my personal experience.

At Netflix, we'd often find that backend services had slow memory leaks, which took a long time to discover and fix because instances rarely lived longer than 48 hours, due to autoscaling policies. If we had chosen to focus on memory leaks instead of autoscaling, Netflix would have been unable to scale to meet demand, and would've been a much smaller business.

Good architecture unifies people, not only systems

A few years later, as a Staff Engineer at a large social media company, I led a project to build analytics for our real-time content personalization system.

The existing design had evolved from a prototype built before I joined. We generated events from our runtime systems that briefly sat in a queue before being exported into an external datastore.

These events captured information such as:

Which content was served to whom at what time

ML model scores for each user/item pair

This data enabled us to:

Track bias across different phases of personalization, from candidate generation to filtering, ranking, and serving

Investigate and debug issues, such as:

Why do some users receive empty pages with zero recommendations?

Why is some content unexpectedly served much more than others?

From a "social" point of view, there was much to consider:

My queue of stakeholders grew exponentially, as everyone suggested two other people to consult.

"We need a design" was a recurring theme among those I talked to. When I asked what goals the current design was preventing us from achieving, I didn't receive useful answers.

There was a prototype for a competing project that a few engineers on the team felt we should use instead of the current system.

The line between what was and wasn't "analytics" was blurry. For example, we were asked multiple times if we could log ML features along with analytics data. We always declined due to a tight schedule, but I felt that we were missing an opportunity to solve a broader, more impactful problem.

We needed a cohesive vision to get everyone on the same wavelength, so I wrote a design document to achieve this which described the problem we were solving - both business-wise and scale-wise - and added a bunch of diagrams. I presented a few options for refining the project and described what I felt was the best path, and why. Then I shared the document.

After a few weeks of conversations with the team and stakeholders, nothing improved and the situation had become notably worse. The team was more fractured than ever, with multiple sub-teams moving in different directions. The notion of a comprehensive analytics solution felt like a distant dream.

I left the company shortly after this, as I no longer felt it was the right fit. But I asked myself for months afterwards what I could have done differently to deliver a better result. I always come back to one answer:

I was too focused on solving a technical problem when the primary issue was a social one.

My job had not been to just make optimal technical decisions. It was to bring everyone together with a unifying vision, whether it was technically optimal or not.

In the video ranking case mentioned above, I lobbied for that project intermittently for over a year before work started: I presented slides and sent them to anyone I could find, gathered a bunch of data, and made more slides. Things went quiet for a while. And then, finally, the stars aligned and we started moving.

In the corporate world, objectively good ideas are rare. Ideas don't have to be *good* per se; they just have to move people forward. Of course, good ideas are more likely to move people than bad ones, but in workplaces, something is "good" when the right people agree it is.

Good architecture balances the practical with the aspirational

Around the year 2020, I was working at a large US TV operator as a Principal Engineer. My role was focused on making broad improvements to personalized video ranking, and I partnered with an Architect to address the problem of making video ranking more flexible. Historically, most content was served by the product either via search, or a variety of human-curated collections. Fully personalizing our content required connecting multiple functions, systems, and teams in a new way.

Let's talk about those teams briefly. Our primary partner was the Search team, a group of 20 or so engineers who operated a high-volume search index, along with a system to access all editorially curated content collections. Their priority was serving relevant search results in a stable, highly available way. A change to their system usually took at least a month to land in production.

We were the Personalization team, a mixture of around 10 Machine Learning Researchers and Software Engineers. We owned several services that provided personalized content to users at runtime, along with numerous data pipelines and random batch jobs. Our priority was increasing user engagement by maximizing the amount of personalized content served. If you asked us to make a change to our system, we could likely get it in production in two business days.

Our goal was to create a proposal with a high chance of winning buy-in from partner teams. One of my self-imposed constraints was that the proposal to increase personalization had to be achievable within our current responsibilities and organizational structure. We needed to increase the speed at which we could put new features and experiments into production, so we had to be mindful and realistic about the demands placed upon the slowest-moving teams.

My architect partner disagreed, saying:

"We should propose the best technical solution. And if that requires organizational changes, let the managers figure it out."

It was an interesting idea, but at the time, a proposal requiring a reorg went nowhere fast at that company. It would also limit my opportunities to write future proposals.

In this case, the Search team moved slowly for legitimate reasons. The Personalization team moved quickly, also for legitimate reasons. One of the goals of our proposed architectural changes was to increase the speed of testing new ML models in production. If adding a new model or configuring an existing model involved manual changes and testing by the Search team, then we would continue to move slowly at their pace. To ignore this constraint would be to provide a proposal of limited value.

A common stereotype of Architects is that they propose things that don't work in the real world. Their solutions require a new, perfect world to be constructed beforehand. In our case, a perfect world could be created by a reorg that breaks apart the Search team, or maybe by rewriting all services from scratch.

But conceiving new worlds isn't the problem; the problem is the inability to intersect a perfect world with the one that actually exists. Sure, maybe a reorg or rewrite would be ideal, but in the meantime, can we deliver something valuable within current constraints to reveal opportunities being missed in the current state?

We were strategically exposed, and mitigated this issue via prototyping. We couldn't win buy-in for a broad proposal, so proposed three different architectures and ran them in parallel with a small amount of real customer traffic. The option with the highest demand on the Search team performed significantly worse than the others, partially due to some architectural flaws, but mostly because they moved more slowly than the Personalization team. Production traffic is the ultimate practical differentiator.

4. Bad architecture is a lot of work that doesn't change much

Let's discuss the Principal Engineer role from the *"Good architecture balances the practical with the aspirational"* section above. I was asked to review a proposal for an "improving personalization architecture" initiative led by an engineer on a team I worked with. It was filled with clearly-stated decisions, such as:

Read more

The Pulse: AWS takes down a good part of the internet

Octob

Monday was an interesting day: Signal stopped working, Slack and Zoom had issues, and most Amazon services were also down. The cause was a 14-hour-long AWS outage in the us-east-1 region. In this issue of The Pulse, we look deeper into what happened and the causes, covering:

Worldwide impact. From Ring cameras, Robinhood, Snapchat, and Duolingo, all the way to Substack - sites and services went down in their thousands.

Unexpected AWS dependencies. Status pages using Atlassian's Statuspage product could not be updated, Eight Sleep mattresses were effectively bricked for users, Postman was unusable, UK taxpayers couldn't access the HMRC portal, and a Premier League game was interrupted.

What caused the outage? A DNS race condition at DynamoDB, Amazon EC2 getting into a congestive collapse, and network propagation issues all stretched the outage to 14 hours.

Why such dependency on us-east-1? It feels like half of the internet is on us-east-1 for its low pricing and high capacity. Meanwhile, some AWS services are themselves dependent on this region.

There's a joke that when aws us-east-1 sneezes, the world catches a cold. It's the most popular region to deploy services in; for many companies it's the default region of choice. It's also the case that aws-us-east-1 is the largest and oldest of AWS's regions and the company itself has several dependencies located there. A us-east-1 region-wide outage usually results in AWS-wide issues which can affect many websites, apps, and infrastructure companies.

1. Worldwide impact

On Monday, 20 October, an outage in this region began at around 3am EST (9am CEST), and lasted a total of 14 hours. Sites and apps impacted:

Read more

San Francisco is back as the world's leading tech hub

Octob

Starting in 2020, the pandemic emptied offices and changed the ways tech companies operate, and how us techies work and collaborate, by making remote and hybrid work patterns the norm. San Francisco (SF) on the US West Coast became emblematic of a major shift: we saw an exodus of tech companies from the city as Pinterest, Stripe, PayPal, and others downsized, Tesla and Oracle moved to Texas, and many VCs, founders, and startups quit for other locations, or simply went fully remote.

But today, things are changing again. I've just returned from San Francisco, where I visited the headquarters of several leading AI companies and up-and-coming startups. Based on my conversations and observations, SF is most definitely back - and it's more important than ever.

A new wave of AI startups has been the catalyst to make SF into a buzzing tech hub once more: there are ever more startups launching in the city, the AI meetup scene is energetic, and it's easy for people to collaborate with one another in person. The tech hub model is proving its value all over again in SF, and might even be one reason for the incredible growth and adoption of AI products and services. *Of course, Silicon Valley has no shortage of innovation, VC funding - and a substantial Big Tech presence - but SF stands out for the number of high-growth startups headquartered there.*

The city looks like a trailblazer for the re-emergence of the tech hub model, reverting back from the remote work surge of 2020-2023. For companies and professionals, this seems relevant - wherever they're based.

Today, we cover:

Cursor: push for release 2.0. A cozy office and a focus on shipping.

AI dev tools startups ahead of the curve. Wispr Flow wants devs to code more by talking, Continue is the "CI/CD infra for AI agents", and Factory builds AI agents for engineering teams.

OpenAI. Scaling faster than any company ever. 10x infra growth for two years straight, with no signs of that slowing down. Refreshingly, OpenAI also hires junior engineers as well as seniors.

Anthropic. Self-organization at scale. Cofounder and former CTO Sam McCandlish shared what helps the company grow, and we discussed their tech stack and engineering challenges.

Buzzing meetups. The local chapter of AI Tinkerers was brimming with "builder energy", and more than the usual amount of insider knowledge about models, maybe because OpenAI and Anthropic are headquartered next door.

Location, location, location. San Francisco-based AI startups seem like they're getting more traction, faster. Location may play a role as it's easy to network, sell to other startups, and hire engineers working on cutting-edge technologies.

AI wave, bubble, or both? It's hard to be certain, but it feels like there hasn't been this rate of explosive growth since the pre-millennium Dotcom Boom.

As always, I have no affiliation with the companies mentioned in this article, and am not paid to write about them. More in my ethics statement.

1. Cursor: push for release 2.0

People swap their shoes for slippers when they arrive at Cursor's headquarters at North Beach, SF, and there was a chill atmosphere during my visit there. That might suggest cosiness, but everyone was focused on the work, with a lot of in-person pairing taking place during what was the final stage of readying Cursor 2.0 for launch. It's going to be their biggest release since the 1.0 launch in June.

An interesting workflow: automatic ticket resolution. Every incoming Linear ticket is automatically assigned to a Cursor agent, which one-shots a suggested fix, and a developer can then decide whether to merge it. This setting is on by default, and devs whom I asked find it very useful.

Cursor's offices: cozy vibe with a focus on shipping

As is common for a dev tools company, the 80 or so developers at Cursor are users of their own product. This makes for an interesting dynamic: software engineer Eric Zakariasson told me that in the last 30 days, engineers there used unique 500K agent sessions and over 110K tabs. Many are now using multiple agents in parallel in their workflows.

Earlier, we covered more on Cursor in the deepdive Real-world engineering challenges: building Cursor.

2. AI devtools startups ahead of the curve

While in the city, I also stopped at the HQ of new startups founded in the last two years which are building platforms that are *already* seeing strong adoption by software engineers.

Wispr Flow: a new modality for programming?

One very interesting startup I stopped by at was Wispr Flow, which creates software to control a computer with the human voice - including writing code with dictation - by using LLM agents. Founded in 2024, the startup found its product-market fit this January, cofounder and CEO Tanay Kothari told me, when the product experienced 90% month-on-month growth which was purely organic without any advertising spend. At the time, the startup saw an impressive 20% free-to-paid conversion rate. Ten months later, this conversion remains unchanged: for every 5 new monthly active users, 1 becomes paid.

In the office, every desk is fitted with a $70 BOYA Gooseneck microphone, into which devs whisper. Despite the coding being done by voice, it was quiet in the office and I couldn't hear Wispr's ML Lead, Menoua Keshishian, while he worked in close vicinity with his microphone.

Wispr Flow's headquarters: ML Lead Menoua Keshishian (left) whispering to Cursor. Bottom right, me with the world's biggest 3D-printed keyboard

Wispr's popularity with devs is due to a few reasons:

Dictation, not transcription. There are countless voice-to-text products that transcribe the spoken word, but this is not useful for coding! For example, if you say: "let's start a server on localhost 3000... no, sorry, localhost 8000", you want the transcript to be able to recognise and cut out the noise, and to read: "let's start a server on localhost:8000".

Dev-friendly touches. When a dev says "localhost 3000", the software automatically generates "localhost:3000." There are many small touches which help with coding use cases.

IDE integrations. Wispr integrates nicely with Cursor, Claude Code, Windsurf, and Visual Studio Code. It's the first dedicated voice-to-text tool to do so.

Whispering *really* works. I was surprised by how well Wispr picks up *quiet* whispering! This - together with the dedicated microphones - makes it feasible even in open-plan offices.

Tanay the CEO told me that they are busy onboarding startups to the product - often by setting up microphones on all desks - and shared an impressive chart (below) showing how devs type less with the tool.

Wispr aggregated user data, based on 20,000+ users. The more people use Wispr, the less they type

The tech stack behind Wispr was interesting to learn about:

Electron app with platform-specific optimizations

Swift: for the Mac and iOS versions. There's no code sharing between them.

C#: for the Windows version

Android: work-in progress, being developed using Kotlin.

One of the biggest technical challenges for the team is the need for low latency: response time must be below 700ms, globally, in order for users to stay engaged. Wispr's servers are US-hosted and must achieve this latency even on high-latency, low-bandwidth networks like 2G.

Today, Wispr's team totals 25 people and is growing fast. You can check out the company here. I'm interested to see how many engineers start talking or whispering to our computers and reject typing, which has been the primary input for coding and software engineering for pretty much as long as those activities have existed.

Continue.dev: the "CI/CD infra" for AI agents inside companies

GitHub started in 2014 at a small office on Howard Street, in San Francisco's South of Market (SoMa) district. The version control company quickly outgrew the place, and today, it's occupied by a new, small and ambitious startup. Continue.dev is a 9-person startup with the mission to become the CI/CD layer for any company using LLMs and agents.

Continue.dev's offices, and some wall art. Center: Brian Douglas, the Head of DX, and me

It's somewhat tricky to fully understand what Continue does because it focuses on companies' internal workflows, and things like what should happen after opening a pull request. They refer to themselves as "continuous AI" that runs background agents.

In pre-AI times, you would have the Continuous Integration (CI) server kick off, run linting, run tests, and report back the results. But with AI agents, there are so many more possibilities! For example:

Summarize the impact of a change

Generate an architecture diagram of the change if it's big enough to warrant one

Verify that API endpoints work correctly if the change touches API endpoints

Generate documentation for the change

Run an evals suite on a change (LLMs evaluate the correctness of changes)

Create a performance analysis of the change

... and more

Yes, it was already possible to hook up an LLM with a CI/CD and do all these changes, but larger companies have more to consider:

Control which models and LLMs are used. You might want to self-host LLMs, optimize for cost or performance, or run parallel LLMs and compare results.

Worry more about security. You want to be very selective about giving LLMs access to credentials like API keys which they might need for testing. Should access even be given?

Logging and auditing. LLMs are nondeterministic, so things sometimes go wrong. You want to have logs and be able to track exactly which agent did what, and when.

I chatted with Continue's Head of DX, Brian Douglas, and he kicked off the background agent for a fix committed to their repository. The agent did the following things:

Summarized the problem and solution

Created an architecture diagram

Verified whether documentation changes were needed

Came up with a merge recommendation and additional suggestions

Analyzed code quality

You can review all the actions on this pull request, which was a "one-shot process" where the agent went off and did a bunch of work based on a single piece of input from an engineer.

Among AI startups, the "996" model for the workday is spreading fast, especially in San Francisco. The name refers to 9am to 9pm work hours, 6 days a week. But Continue is showing that a punishing work pattern is not the only way to succeed. During my visit, the office was largely empty at 5:30pm. Everyone works hard while they're there, Brian said:

"If we work silly long hours, it's mainly because we are already excited about the solution we are shipping, not because there's some unwritten expectation to stay late at the office. At Continue, the emphasis is on results, not desk time. Some of the team take advantage of hybrid days to do focused work".

Continue already counts major AI labs and Big Tech companies among its customers: the ones that set up these LLM-powered CI/CD workflows. Continue.dev is a good example of how much a small team can achieve by working hard and smart. *Learn more about them by checking out these MCP "cookbooks" showing how the platform works. Best of luck to the team!*

Factory AI: AI agents and enterprise teams

I tend to think of Factory.ai as a Claude Code for enterprises, and for supporting all major LLMs. They call their agents "droids", which can be kicked off to do work asynchronously from an IDE, Slack, or the command line. Setting up Factory is as easy as getting started with Claude Code: just run a curl command, and your workspace is up and running.

Recent data from AI startup Modu ranked Factory as #2 in pull requests approved (with 75% of PRs raised by droids approved), slightly ahead of OpenAI's Codex and Anthropic's Claude Code.

I met cofounder and CEO Matan Grinberg and the core team at 6pm, when the office was half full.

SuperFit

Factory's HQ: the office is next door to Wispr Flow's, making collaboration between AI startups easy

While many dev tools companies focus on individual engineers' productivity, Factory focuses on accelerating teams.

The team digitalizes everything they do and feed it into their product. I found some details particularly interesting:

Every meeting is transcribed

All chalkboard jams are photographed and uploaded to the backend

Design docs are not done by hand, they are kicked off with Droids using all context uploaded to Factory's own systems

Their engineering team is exploring - and blogging about - "agent native development", where software engineers always leverage AI agents as the first step of their day-to-day workflows. An interesting thing the team found while working with engineering customers was that the "quality" of an AI agent is less important than following best practices that help agents (and humans!) to build better software. These include:

Having good specifications

Having automated tests, and writing them as functionality is added

Mandating a way to "verify" each pull request before putting it up for review

I like how the Factory team is open-minded in exploring new ways of building *quality* software. Naturally, their experiments focus on AI agents, but my sense is that they're finding that good software engineering practices result in higher quality code, *especially* when working with AI agents. *You can read more about the Factory team's learning in their engineering blog. Check out the company here.*

3. OpenAI: scaling faster than any company ever

OpenAI's offices are large, but the corporate branding that's common in many workplaces is largely absent, with very few logos on display. One possible reason OpenAI executes faster than most startups is its in-person work culture that has existed since its early days, as we covered in a 2023 deepdive into OpenAI's engineering culture.

OpenAI's offices, and swag

During my visit, I met with Sulman Choudhry, who heads up ChatGPT's Engineering and Applied Research functions. He had a few interesting things to say:

OpenAI now hires junior engineers, not just seniors. The company is finding success with a "super senior + super junior" mix. Super juniors are early-career software engineers, who are AI-native, entrepreneurial, and several of whom were at the startup accelerator Y Combinator, which is also headquartered in San Francisco. Super juniors use AI tools in ways that surprise more experienced colleagues, Sulman said.

For example, one super junior did an impressive piece of work, and Sulman asked if he'd used Codex for it. This prompted a defensive response, which Sulman thought might signify that the junior had hand-coded most of it and resented the suggestion that they'd simply delegated the task to AI. Sulman recounted the engineer's reply:

"It's a little insulting that you ask if I used Codex for it. Of course, Codex by itself couldn't build software of this complexity, and you probably also know it.

It's why I had to use *many* instances of Codexes to build it, setting up communication channels between instances, so they can take on the task, one of them validating it, and others implementing the specialized parts."

So, the engineer was *not* annoyed by the assumption they'd used Codex; they were irked that it was assumed they'd employed only a *single* instance of it! *Come on, who would want to slow themselves down so much by doing that?!* Super juniors at OpenAI are pushing the limits of AI tools, and inspiring more experienced engineers to use them in new ways.

Weekly internal ChatGPT demos

Unsurprisingly, every engineer and team builds custom tools, and there's an explosion of MCP servers connected to Codex. Each week, teams demo the tools they build and share things that work for them, and how they are changing their workflows.

A recent interesting demo was a "fix this" button integrated into the internal version of the OpenAI mobile app. We're all familiar with the "report a bug" button in mobile apps: it takes a screenshot, you describe what's wrong, and then submit it. A team at OpenAI took this a step further: you fill out the bug report, and then tap a "fix it" button. The bug report goes to Codex and a fix is drafted. An engineer needs to approve the fix - but the feedback loop is much faster!

That demo was so successful that the "fix it" button is now part of ChatGPT.

Scaling faster than everyone

To learn about infrastructure challenges, I met Emma Tang, who heads up OpenAI's Data Platform team which owns four main areas:

Big data infrastructure: operating things like a Data Lake (a centralized repository storing massive amounts of raw data), Spark (distributed computing framework to process large-scale data), Trino (formerly known as PrestoSQL, a distributed SQL query engine), Airflow (workflow orchestration), business intelligence tools and dashboards.

Real-time streaming: supporting infrastructure platforms operating at scale like Kafka (distributed event streaming and message broker), Flink (stream processing), and an internal system called K-Forwarder.

ML infrastructure: non-LLM product features, recommendation services, and a Chronon feature store (an open source feature platform for ML engineering created by Airbnb.)

"Flywheel team": a team unique to OpenAI that's responsible for production data scrubbing, removing all personal identifiable information (PII), and preparing training data.

Compute demand for the infra layer has grown 10x annually for the past two years. The compute for the fleet serving the infrastructure and data lake storage has vastly expanded, and there's no sign of that slowing down. Emma discussed how growing so fast creates engineering challenges:

Building for scalability without needing to rewrite the system, even as it grew to ~100x scale in 2 years. Scaling is done horizontally: the team went from having 1x Kafka cluster to having 5 today, and from 1x Airflow instances to 4.

Hitting technological limits: for example, encountering row size limits in Spark data frames from storing lengthy chat conversations.

Adding support for multimodal features across ChatGPT products involves storing much more and richer data, such as voice data, images, and PDFs.

The data infrastructure team is very senior: around 95% of members are data infrastructure specialists, and a third are members of the Apache Project Management Committee (PMC). Spark, Airflow, Kafka, and Flink are all Apache projects, and a third of the team are committers to at least one of them.

Fundamental rewrites might be unavoidable. Even though the team prepares for as much growth as possible, the next 10x tranche of growth might require major rewrites, since it's hard to keep patching a system that's achieving such magnitudes of growth (around 1,000x) without rethinking some basics. Needless to say, a rewrite would involve increasingly challenging migrations.

Personal reflections on OpenAI: The engineers I met are motivated, focused, and at the top of their game. The scale of engineering challenges they face are rare, and engineers get unusually high autonomy to solve difficult problems. There is no lack of ambition, nor signs of stalling growth.

We cover more about OpenAI in these deepdives:

Building, launching, and scaling ChatGPT Images

Inside OpenAI: how does ChatGPT ship so quickly?

Scaling ChatGPT: five real-world engineering challenges

4. Anthropic: self-organization at scale

At Anthropic, I sat down with cofounder Sam McCandlish, who was until very recently the CTO. Sam is a quantum gravity physicist by education who worked at OpenAI, and then joined Anthropic's founding team. Since then, he's helped the company grow from a workforce of 7 to more than 2,000 today. Over the last 4 years, Sam has committed code to pretty much every system across the company, and has worked in research and engineering. After onboarding Anthropic's new CTO, Rahul Patil (formerly CTO at Stripe), he plans to return to solving problems in large-scale model training. I found him thoughtful, open, and humble; someone who thinks from first principles.

Anthropic's rapid growth challenges

I asked Sam how he approached the company's breakneck speed of growth. He made a few points:

Read more

Beyond Vibe Coding with Addy Osmani

Octob

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. Graphite is one example of a company that has built granular control and rapid iteration into their development workflow using Statsig. Read more about how Graphite uses Statsig and try it out.

?? **Linear** - The system for modern product development. Vibe coding is having its moment: but what vibe coding doesn't account for is institutional knowledge. Linear agents, on the other hand, work inside your team's shared workspace. See how agents work in Linear and give Linear a go to try it yourself.

-

In this episode

Addy Osmani is Head of Chrome Developer Experience at Google, where he leads teams focused on improving performance, tooling, and the overall developer experience for building on the web. If you've ever opened Chrome's Developer Tools bar, you've definitely used features Addy has built. He's also the author of several books, including his latest, *Beyond Vibe Coding,* which explores how AI is changing software development.

In this episode of *The Pragmatic Engineer*, I sit down with Addy to discuss how AI is reshaping software engineering workflows, the tradeoffs between speed and quality, and why understanding generated code remains critical. We dive into his article *The 70% Problem*, which explains why AI tools accelerate development but struggle with the final 30% of software quality-and why this last 30% is tackled easily by software engineers who understand how the system *actually* works.

Jump to interesting parts

The 70% Problem (**22:15**)

Why you should keep expectations low, and control high when using AI coding tools (**32:58**)

How the engineering manager and product manager roles might be changing with these AI tools (**42:49**)

An interesting quote from this episode

How Addy uses AI tools for his own coding:

Gergely (at 06:14): "How do you personally use these AI tools?"

Addy: "**I have more recently been focusing on the idea of spec-driven development,** having a very clear plan of what it is that I want to build.

There are definitely places where I still vibe code. If it's for a personal tool, something throw away, in the old days, if an engineer or a PM had an idea, maybe we would put together a quick mock, maybe we would put together a wireframe or a sketch or something like that. Perhaps work with UX to come up with something a little bit more polished.

I think that tests are great. And if you are open to tests, it can be a great way of de-risking your use of LLMs and coding. Because sometimes, even if you're using state-of-the-art models, you can end up in a situation where maybe the code looks more convoluted than you would expect. Or maybe your first couple of prompts have been generating really good code, and then for whatever reason, things go off. the rails. But if you can prove that things are working with tests and that if something did go off the rails, it's clearer to you what that was, I think that that can help you keep your project

green the whole time."

The Pragmatic Engineer deepdives relevant for this episode

Vibe Coding as a software engineer

How AI-assisted coding will change software engineering: hard truths

AI Engineering in the real world

The AI Engineering stack

How Claude Code is built

Timestamps

(00:00) Intro

(02:17) Vibe coding vs. AI-assisted engineering

(06:07) How Addy uses AI tools

(13:10) Addy's learnings about applying AI for development

(18:47) Addy's favorite tools

(22:15) The 70% Problem

(28:15) Tactics for efficient LLM usage

(32:58) How AI tools evolved

(34:29) The case for keeping expectations low and control high

(38:05) Autonomous agents and working with them

(42:49) How the EM and PM role changes with AI

(47:14) The rise of new roles and shifts in developer education

(48:11) The importance of critical thinking when working with AI

(54:08) LLMs as a tool for learning

(1:03:50) Rapid questions

References

Where to find Addy Osmani:

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

? LinkedIn: https://www.linkedin.com/in/addyosmani

? Website: https://addyosmani.com

? Newsletter:

Mentions during the episode:

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

? *Building large scale web apps*: https://www.amazon.com/Building-large-scale-web-apps-ebook/dp/B0CZ9H2XN5

? Chrome DevTools (MCP) for your AI agent: https://developer.chrome.com/blog/chrome-devtools-mcp

? Mitchell Hashimoto on X: https://x.com/mitchellh

? Cline: https://cline.bot

? VS Code: https://code.visualstudio.com

? Cursor: https://cursor.com/

? Copilot: https://github.com/features/copilot

? The 70% problem: Hard truths about AI-assisted coding:

ElevateThe 70% problem: Hard truths about AI-assisted codingAfter spending the last few years embedded in AI-assisted development, I've noticed a fascinating pattern. While engineers report being dramatically more productive with AI, the actual software we use daily doesn't seem like it's getting noticeably better. What's going on here...Read morea year ago · 1478 likes · 74 comments · Addy Osmani

? How AI-assisted coding will change software engineering: hard truths: https://newsletter.pragmaticengineer.com/p/how-ai-will-change-software-engineering

? Python, Go, Rust, TypeScript and AI with Armin Ronacher: https://newsletter.pragmaticengineer.com/p/python-go-rust-typescript-and-ai

? DeepMind: https://deepmind.google

? Gemini: https://gemini.google.com

? Stackoverflow: https://stackoverflow.com

? Jules: https://jules.google

? CodeX: https://chatgpt.com/codex

? JavaScript: https://www.javascript.com

? Bolt: https://bolt.new

? *Building Web Apps with Bolt: Unlock the Future of AI-First Development*: https://www.amazon.com/Building-Web-Apps-Bolt-Development/dp/B0F6SP872Y

? *The Software Engineer's Guidebook: Navigating senior, tech lead, and staff engineer positions at tech companies and startups*: https://www.amazon.com/dp/908338182X

? *AI Engineering: Building Applications with Foundation Models*: https://www.amazon.com/AI-Engineering-Building-Applications-Foundation/dp/1098166302

? AI Engineering with Chip Huyen: https://newsletter.pragmaticengineer.com/p/ai-engineering-with-chip-huyen

? The AI Engineering Stack: https://newsletter.pragmaticengineer.com/p/the-ai-engineering-stack

-

Production and marketing by Pen Name.

The Pulse #151: Amazon layoffs - AI or economy to blame?

Octob

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:

Amazon layoffs: AI or economy to blame? Amazon is doing more mass layoffs, claiming it wants to be more nimble. But are job losses really about US economic fears, and how Amazon's retail business will be affected?

Industry pulse. Meta lays off 600 people from its AI division while hiring for another AI unit, $100K H1B fee will not apply to new grads, more AWS and Azure cloud outages, DEI costs the Python Software Foundation $1.5M, Citi blocks premium customers for weeks, and more.

OpenAI becomes for-profit, and takes on Chrome. Years overdue, OpenAI has finally changed its org structure to the for-profit model it's been operating as. It has also launched Atlas, an AI browser meant to take on Chrome.

NVIDIA the biggest in Big Tech by a distance. NVIDIA is worth as much as Google and Meta combined, and is starting to expand from "just" selling GPUs to direct relationships with large companies such as Palantir, Eli Lilly, and Nokia.

1. Amazon layoffs: AI or economy to blame?

Online retail giant Amazon unexpectedly announced 14,000 job cuts earlier this week. The massive round of layoffs at the company follows other mass redundancies in recent years:

Read more

Tech jobs market 2025, part 3: job seekers' stories

Nover

"What is the state of the tech jobs market in 2025?" is the question this article tackles in the third and final part of our mini-series on that major subject. We hear from job platforms, and from tech professionals searching for their next opportunity. This article features Wellfound (a jobs platform with around 6M software engineering profiles), and data from Revealera, an alternative data platform. There are also more than 30 software engineers and engineering leaders who discussed their job hunting experience with me. Today, we cover:

Job platform data. Falling demand for remote work, more "barriers" put up by companies, slightly higher demand for backend engineers than before, and more.

Junior engineer recruitment rebounds? More scaleups and publicly traded tech companies are doubling down on hiring new grads and early-career engineers.

Picky employers. More hiring managers hold out for the "perfect candidate," more rejections come with no feedback, and some think the candidate quality is down, overall. Referrals seem like the only way to consistently get interviews.

Software engineer archetypes in and out of demand. AI engineers, those with Big Tech experience, and infra+SRE engineers are in demand. Times are reportedly tougher for candidates who took career breaks, are self-taught, or are native mobile engineers.

State of engineering leadership hiring. It's very tough everywhere, especially for experienced engineering managers not yet at Director level. One executive recruiter says many leadership candidates have poor AI skills and unrealistic pay expectations.

Remote market. It's harder to land a job and compensation is lower, but the bar is higher for remote positions.

Regional observations. Outside of cities, the market is tougher: in Germany, Wayfair's exit could drag down pay. Are Swiss companies looking to hire in cheaper EU countries?

Previous articles in this series covered:

Part 1: what the data says

Tech job stats

AI Engineering trends

Big Tech hiring stats

Growing importance of location

Tenure rising fast at Big Tech

Where are Big Tech engineers moving to and from?

Engineering leadership recruitment

Remote jobs

Part 2: what hiring managers see

Flood of applications

Few hires via inbound

'Top' candidates are hard to find

Remote jobs: more competition for less comp?

Fake applicants + AI: a growing problem

Higher demand for founding engineers and product engineers

Early-stage startups have their own hiring problems

As usual, I have no affiliation with vendors mentioned in this article, and have not been paid to write about them. More in my ethics statement.

1. Job platform data

In Part 2 of this mini-series, we covered how some hiring managers are shifting to recruit on sites like **Wellfound** (formerly: Angellist Talent) because LinkedIn has gotten too noisy for finding candidates efficiently.

Wellfound has **12M active candidates** on its site, of which circa 50% are software engineers. More than **27,000 companies** use the site for recruitment, with more than 100,000 hires made, to date. As a jobs marketplace, Wellfound does not filter. Its CEO Amit Matani has shared the trends they see, right now:

According to Wellfound's data, the job market was most competitive in January 2025, when applications per job peaked on the site. Today, the number is down 10% from that height - meaning it's now slightly easier to stand out as a candidate. Even so, the job market is very tough compared to 2021:

6x as many applicants per remote job

5x as many applicants per in-person job

There's a big rise in AI engineering job ads. Overall, this is the most visible trend on the platform, and it matches trends covered in Part 1 of this mini-series. *For advice on getting into AI Engineering, check out the deepdives, AI Engineering in the real world and The AI Engineering stack.*

What makes an in-demand candidate?

I asked Amit about profiles of candidates which are most popular with employers, meaning types of folks whom companies reach out to the most. He revealed:

AI product engineers. Those who can build AI products end-to-end (fullstack + LLMs + evals + RAG).

Data engineers and backend engineers. Visibly rising demand for these.

AI-native engineers: Engineers who are experts in how to use AI tools to significantly boost productivity.

Pedigree means up to 50x more reachouts. Profiles with high-profile schools and workplaces get up to 20-50x more reachouts than similar profiles with not as much "pedigree". This is not unique to software engineering, says Amit, nor to 2025.

In-person much more in demand. Most hiring is by companies headquartered in US metropolitan centers, and engineers in such areas get many more reachouts than candidates who work full-remote.

The San Francisco Bay Area has the most job postings - not even a contest. Overall, Wellfound finds the highest density of job ads in the US and India.

Remote: falling demand, but many roles are open to it

On remote job trends, Amit shares:

"35% of engineering jobs are open to remote candidates. This is down from a peak of 56% in 2022. However, remote engineering jobs get 4.5x as many applications compared to 2022."

Visualizing this:

% of jobs open to remote applicants

The remote jobs market is much more competitive than in 2022. Source: Wellfound

Hiring trends: candidates face more barriers

Below are some trends among employers from Wellfound's data:

Hiring pipeline trends:

Inbound: noisier. This still drives some hires for all companies, but has become the noisiest channel of all. Employers face more work combing through large numbers of inbound applications.

Most companies focus on sourcing. They reach out to potential candidates directly, instead of waiting on inbound applications.

Screening trends:

More barriers for job seekers. Inbound is noisy because there's lots of applications, but most are under-qualified. So, companies place barriers like asking additional questions upfront, scheduling async calls, assigning coding assessments - and even doing AI interviews in advance.

AI tools for ranking and filtering candidates. To deal with high volumes of inbound applications, AI tools are used to filter out applicants who are likely to be fakers, and to create an order of priority for identifying strong inbound applications. There's still human review, and no such thing as "AI rejection", Amit told me.

Interview trends:

More in-person interviews and lengthier takehome exercises.

Trial days or weeks, pre-offer. More companies use a trial period before making an offer. This could be a day or week where candidates work with the team. It's pretty similar to the paid "trial week" concept at Linear.

AI is being integrated into interviews. Companies have accepted there will be AI in interviews, and are changing their processes to accommodate it.

Fake jobs, fake candidates

I asked Amit about two topics getting plenty of attention in online forums: the suspicion that some job ads are fake (meaning no hire is made and the sole purpose is to collect resumes), and that some candidates are "fake" because they use made-up CVs and hope to get hired to jobs they're totally unqualified for.

Fake jobs: not a visible trend - but "ghosting" is. Amit shares:

"Posting fake jobs with the sole intent of capturing resumes is not widespread - at least not on our platform, or at companies I talk to. That said, companies not responding to candidates is a problem. Candidates often don't hear back because:

Hiring managers / recruiters are indecisive or unprepared. They might have put up a role without really being ready to hire. Or they might not be sure about a candidate, but also don't want to "close the door" by sending a rejection.

Hiring managers are overwhelmed. Some companies we work with get 1,000 applicants a week for a single role! It's just hard to process that.

There really isn't much of an incentive for recruiters to send a rejection email.

We try to incentivize following up: we give companies 2 weeks to accept or reject applications. We find that if the company does not respond within 2 weeks, the likelihood of them sending a followup is extremely low."

Fake candidates are an ongoing problem. Amit:

"Fake applicants are a problem and there are three types:

Scammers or state actors who are impersonating candidates in order to deceive employers.

"Auto-appliers" who aren't interested in a role and are a poor fit, but apply anyway.

Embellished profiles: candidates who add misleading information to their profiles; for example, stating they're US-based

when actually in India.

The biggest issue we're seeing is scam candidates. It's an issue for sourcing and applications. These candidates create Linkedin and Wellfound profiles, and apply to jobs. They also stuff their profiles with great companies and in-demand keywords so they show up when companies do sourcing. We make a lot of effort to keep such candidates off our platform with automated detection, reporting, and ID checks.

Auto-appliers are the next biggest thing: candidates who apply and are not good fits, overwhelm companies. A lot of them then ghost employers, even when one responds".

More hiring trends from Wellfound

Additional observations:

Onsite hiring is dominant, but companies hiring remotely have the upper hand. Practically every startup Wellfound works with wants candidates to work onsite. But many candidates want the opportunity to work remotely, at least partially. Companies that hire remotely definitely have the upper hand in terms of talent choice.

There are AI budgets for recruitment. Many teams have "AI budgets" to try out new things with. So, there is a decent amount of experimentation in AI recruitment tools.

It's an employer's market. Companies know this and act accordingly: they are more picky, and put up more barriers.

Auto-appliers are growing. More candidates use auto-apply software to apply for hundreds or thousands of jobs. It's a vicious cycle: with more inbound applications, fewer applicants hear back, and so more of them use auto-appliers.

Highest demand for software engineers with 3-15 years of experience, willing to work onsite. Amit shares that the most "promising" profile they see is for a software engineer with a few years of experience, based in locations where many companies hire. *Location has suddenly become a lot more important, as reported in Part 1 of this mini-series.*

Shifting demand for software engineering specializations

Revealera is a data provider of job postings for financial companies. The CTO, Henley Wing Chiu, built a machine learning model to analyze 180M job listings from the Revealera platform collected in the past 12 months. He detected trends in the subset of software engineering recruitment and has shared the findings in this article:

Source: [Revealera/Henley Wing Chiu]

This chart tells us a few things:

Possible shift to fullstack engineering. The decrease in frontend engineer positions could be because of a shift to hiring fullstack engineers, instead of purely frontend engineers.

Ongoing drop in demand for native mobile engineers. This started in 2022, *as covered in [Is there a drop in native iOS and Android hiring at startups?]*

More demand for AI Engineering and Data Engineering. This is in-line with other reports, and with what we found in Part 1. Check out a deepdive on data engineering.

2. Junior engineer hiring to rebound?

One unfortunate trend since 2020 is that hiring for entry-level software engineering roles seems to have almost stopped, industry-wide:

In 2020-2022, most companies started to work remotely out of necessity due to pandemic lockdowns. Onboarding junior engineers was tough in this situation, so companies delayed hiring them.

In 2023-2024, most businesses used a hybrid setup. However, the layoff wave of 2022 increased the supply of highly skilled senior+ software engineers who used to be hard to hire. As a result, many companies hired them.

One outcome has been a very tough period for new grads and those with little professional software engineering

experience. Now, there are signs that junior-level hiring may finally be returning:

OpenAI and Anthropic are hiring junior software engineers for the first time ever. I confirmed this with engineering leaders at each place while visiting their respective headquarters in San Francisco, a few weeks ago.

Shopify, GitHub, Cloudflare: uptick in internships. Shopify is hiring 1,000 interns per year, Cloudflare will be onboarding 1,111 interns in 2026 (the number is a nod to its 1.1.1.1 DNS address), and GitHub is hiring more interns than before. Only a subset of all interns convert into full-time employees, and Shopify has extended full-time offers to around 1 in 5 interns, so far. These companies have never invested as much in interns as now.

Netflix: the streaming giant only hired senior software engineers for 25 years, but has started to onboard new grads recently, as I learned by talking with Netflix CTO, Elizabeth Stone.

Below are anecdotes from hiring managers which suggest junior hiring could be picking up, based on my conversations with them:

Publicly traded tech companies and scaleups

"**After two years, we're hiring juniors again.** Our company pushed AI to its limit, where people were allowed to push 10K lines of code PRs that were barely reviewed. This has caused an unmaintainable mess at the foundational layer. So my company did not hire junior engineers for the past two years. Now they are walking back from it." - Engineering Manager, publicly traded company, San Francisco.

"**We've started to hire juniors after 3 years.** We paused junior hiring about 3 years ago, and in the summer of 2025, we still had no intern class. But the good news is that we're targeting to hire 5-6 juniors by 2026, to bring them into a 2-year rotational path, and to get them to mid-level. We've started interviewing new grads more, as well." - Director of Engineering, scaleup, New York.

"**We tend to fill junior and graduate positions more easily than experienced ones.** There is a larger pool of junior and graduate engineers." - Director of Engineering, Fintech, London.

Even Amazon seems to be hiring more new grads, according to a Software Developer Manager (SDM) there in Seattle:

"For all the talk about AI replacing entry-level jobs, it feels like the trend is the opposite. We are seeing a large influx of L4s [Amazon's entry-level role] from good schools. These new grads say most of their friends have got into Meta, Google, or similar places. There are not many left without a job".

Pivot to hiring new grads with AI fluency

An Engineering Manager in Boston at a publicly traded company said his business is pivoting to onboarding many more junior engineers:

"We're pivoting to hiring junior and fresh-out-of-college talent. Then, where necessary, we are hiring folks with a focus on highly scalable systems, who come from Big Tech. We're looking to fill most open roles with very junior talent for different reasons:

Anti anti-AI bias. We'd never hire an engineer with a whiff of anti-AI bias, these days. Junior engineers are very pro-AI.

We convert the best interns to fulltime engineers. We have a very strict intern hiring approach that is not very forgiving. It has its ups and downs: we get to hire the best of the intern group, but it's very time-consuming to give thorough feedback for everyone.

We don't need Big Scale folks for many roles. Being hands-on with AI tools is often more important, so why hire big-scale people?

We're investing in the bottom rung of the ladder. Junior engineers bring a lot of energy and creativity, and complement senior-heavy teams like our company has historically been."

The same place is hiring "AI infra" engineers, the Engineering Manager revealed:

"We are investing in 'AI infra' with experienced engineers. We're specifically hiring for AI engineers to embed SRE style to try and raise the benchmark of AI adoption in key areas.

There's a lot of teeth gnashing about the death of the junior engineer, but we're doing the opposite. I agree with the direction, and it seems to be working well, so far."

Still tough for juniors - Wellfound

Data from Wellfound suggests the market remains challenging for junior profiles. Amit says:

"Junior folks are having a harder time. There are fewer jobs out there, and just a lot more computer science grads on the market compared to years ago. The junior characteristics that get reachouts tend to be:

Pedigree. Profiles from top schools or 'brand name' work experience

Practical projects. They have projects which clearly demonstrate they can build production-ready software.

AI native. Profiles that lean into being AI-native and use AI tools better. I know several companies that hire a few really senior people and a bunch of 'AI-native juniors' to pair with them".

3. Picky employers

One of the most-mentioned observations is that companies are more selective with their hiring, compared to recent years. For a start, it's hard to get responses even when candidates *reply* to inbound messages from recruiters. Here's a software engineer with 13 years' experience from Portugal, who recently worked at a crypto startup. He described the experience as "very weird":

"**90% of the time, I'm 'ghosted' when responding to LinkedIn recruiter messages.** When I ask for more details about things like tech stack, location, if the role is remote or hybrid I don't get any answer 90% of the time!

This experience is very strange: I never had trouble changing companies in the past, and 'ghosting' was not something I've experienced before.

To add insult to injury, when I did get to a first interview, I got rejected. This never happened in past job searches."

Even for engineers who get lots of inbound messages, securing an offer is tough. A frontend engineer with 25 years of experience, based in California, shares:

"**Hiring managers seem to be looking for the "perfect person".** I get plenty of inbound from recruiters. However, I still feel that expectations are up and salaries are down. It feels like hiring managers are in a rush to reject and not in a rush to fill positions".

More observations by candidates:

Companies are more cautious. They seem to focus on very specific, specialized areas.

Large companies are not really hiring from startups. A software engineer in the US with 20 years of experience, and a background of working solely at startups, found very little success with reachouts. Meanwhile, his wife, a software engineer with Big Tech experience, gets far more interviews. A hiring manager said that he was the only candidate in the loop without experience at a large company, which made him feel "like showing up in crocs to a country club." His conclusion: "I've felt like companies are just hoping to grab recently laid-off Big Tech workers at a discount."

Referrals are more important than ever. Several engineers shared that their current company prioritizes referrals when fast-tracking interviews because these are more trustworthy. Meanwhile, inbound applications are stuffed with keywords, and too many unqualified candidates apply.

Long interview processes with no feedback are more common. UK-based software engineer Sam Rose shared in this thread how, after 18 hours of interviewing, he received a generic rejection and no feedback, despite going through the full interview loop.

'Weaker' candidates?

I asked a hiring manager who's also looking for a job why he thinks employers are stricter about hiring. They told me they think companies are not harsher, but that candidates aren't as strong as before:

"Candidates for our senior engineering position have been universally underwhelming. The people I've interviewed are either:

Well underqualified for a senior position

At a senior level in Big Tech, but an interview reveals massive skill gaps

For example, a fullstack engineer with 10 years at a big-name software company had zero experience with frontend styling systems. All engineering complexity was taken care of by a frontend platform team, or a dedicated infra team. This engineer had no in-depth knowledge of frontend systems or database internals".

The view that candidates are weaker than before is echoed by several software engineers involved with hiring at their workplaces. *Or maybe expectations have gone up!*

Employers like referrals & ignore inbound applications

An experienced software engineer in the UK shares:

"Good devs with proven delivery records are approached directly, by recommendation. Lots of offers, many I'm receiving are 'we will pay what's necessary to get you'. I spoke with a data company last week where one of my former coworkers works, which is hiring Python devs. They received lots of CVs for inbound applications, but they are not interviewing those applicants.

In the words of the CEO: many CVs look perfect, but they can't trust inbound candidates anymore. They would rather pay more and 'poach' someone they previously worked with, than pay market rate and take risks with untrustworthy candidates. So they are pinging the best engineers their team recommends and reaching out, one by one.

Recruiters working with companies are sending over multiple candidate profiles, but strongly recommend devs they worked with in the past, who they think are strong.

Trust is now the number one criterion in the UK market. For my current job, I was hired after a 15-minute chat about a project, about which I spoke barely a few sentences. This was all at a place where an engineer previously worked with me".

Personally, I don't think *most* companies completely ignore inbound applications, but there is a strong shift towards referrals. Every "fake candidate" caught in the application process will reinforce to hiring managers that they should focus on referrals and sourcing candidates directly.

Hiring dynamics during a recruitment freeze

An engineering manager at a 4,000-person company in the Netherlands shared how their company has had a hiring freeze for a year.

The company is listing "ghost jobs" due to internal politics. The company *does* have open positions advertised on its website, but these positions do *not* get approved for external hiring, even after 3 months. In reality, these are "ghost jobs", and a bottomless pit for applicants.

This engineering manager described how the hiring freeze affects things:

Poaching from other teams is the new norm. For ICs who are high performers, there is a lot of politics, and it's like Game of Thrones to poach from other teams because they can't hire externally.

"Mobbing" up to push out ICs is common. EMs, Lead devs, and other management use mobbing to push out ICs they dislike. Since ICs are hesitant to look for a new job in the current job market, they look for new jobs internally. Eventually, another team hires them and they switch, meaning the externally-advertised position is filled.

Internal poaching is very important for sticking to budgets. Every team and manager is given a budget which doesn't leave the company. If you can backfill from another team, the cost of this headcount is added to your cost center. If someone leaves your team and you cannot "poach" to backfill, then your budget will be lower next year. Poaching internally is important for those who want to be within budget by the end of the fiscal year.

Contractor rates are dropping. Teams hire contractors (freelancers) as a last resort. There's less hiring than before, and the pay is slightly lower. Also, contract lengths used to be 12+ months - but now are no longer than 6 months due to budget uncertainty.

4. Software engineer archetypes in and out of demand

Even in the current market, certain types of engineer are *incredibly* in demand:

Read more

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** ? - ? 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.

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...

Read more

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 O
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	
↓	
First draft	Dev
↓	

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 presentat
      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 caree
      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 into 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.

Email Address

Get notified

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

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

New Report Edit Criteria

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!

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 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.

Read more

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

L5

L6

L7

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):

Read more

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.

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.**

Read more

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)

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.