The Software Engineer's Guidebook

Navigating senior, tech lead, and staff engineer positions at tech companies and startups

Gergely Orosz

The Pragmatic Engineer

SHROFF PUBLISHERS & DISTRIBUTORS PVT. LTD
Mumbai Bangalore Kolkata New Delhi

The Software Engineer's Guidebook

by Gergely Orosz

 ISBN: 9789083381824

Published by Pragmatic Engineer BV, Amsterdam, Netherlands.
Edited by Dominic Gover

First Edition: November 2023

First Indian Reprint: November 2023

ISBN: 978-93-5542-471-6

Published by **Shroff Publishers and Distributors Pvt. Ltd.** B-103, Railway Commercial Complex, Sector 3, Sanpada (E), Navi Mumbai 400705 • TEL: (91 22) 4158 4158 • FAX: (91 22) 4158 4141
E-mail : spdorders@shroffpublishers.com • Web : www.shroffpublishers.com Printed at SAP Print Solutions Pvt. Ltd., Mumbai.

Keep up-to-date with the tech industry

This book has taken four years to write, and within its pages, I try to capture observations and advice that will stand the test of time. This means that changeable parts of the tech industry – like the job market, levels of funding for startups, and emerging technologies – are not within the scope of this book. This is despite the fact I write about these timely matters every week, as the author of The Pragmatic Engineer Newsletter.

The Pragmatic Engineer Newsletter is one of the most-read online technology publications and the #1 technology newsletter on Substack. It's a weekly newsletter offering an inside look at Big Tech and startups, with its finger on the pulse of the tech market. The newsletter is highly relevant for software engineers, engineering managers, and anyone working in tech.

Bloomberg describes my newsletter like this[1]:

> "In his newsletters, [Gergely Orosz] speaks with inside knowledge and authenticity about the industry. The Pragmatic Engineer covers a lot of ground, from sweepy pieces about the state of Big Tech and layoffs in the industry, to how employees should prepare for performance reviews."

Subscribe here: www.pragmaticengineer.com

I hope you enjoy this book, and find it useful for keeping up-to-date with the tech industry, together with the newsletter.

– Gergely

[1] https://www.bloomberg.com/news/newsletters/2022-11-25/top-tech-newsletter-on-substack-is-written-by-an-engineer-who-gets-scoops

Contents

PREFACE

I've been a software engineer for about 10 years, and a manager for five more. During my first few years as a developer, I received little to no professional guidance. But I didn't mind, as I assumed hard work would eventually lead to progress.

However, this changed a few years into my career when I was passed over for a promotion to a senior engineer role which I thought I was ready for. Not only that but when I asked my manager how I could get to that next level, they didn't have any specific feedback. It was then that I decided that if I ever did become a manager, I'd always offer team members useful advice on how to grow.

It was when I was working at the riding-hailing app Uber that I became an engineering manager. By then I was a seasoned engineer, but I still remembered my earlier promise to myself. So, I did my best to support people on my team to improve professionally, get the promotions they deserved, and give clear, actionable feedback when I thought colleagues weren't ready for the next level, just yet.

As my team grew and I took on skip-level reports, I had less and less time to mentor teammates in-depth. I also started to see patterns in the feedback I gave, so began to publish blog posts of the advice I found myself giving repeatedly; about writing well, and doing good code reviews. These posts were warmly received, and a lot more people than I expected read and shared them with colleagues. This is when I began writing this book.

By year two of the writing process, I had a draft that could be ready to publish. However, at that time I launched The Pragmatic Engineer Newsletter. The focus of this newsletter is keeping the pulse of today's tech market, plus regular deepdives into how well-known, international companies operate, software engineering trends, and occasional interviews with interesting tech people. Writing the newsletter made me realize just how many "gaps" were in the book draft. The past two years have been spent rewriting and honing its contents, one chapter at a time.

After four years of writing, I can say with conviction that "The Software Engineer's Guidebook"

and The Pragmatic Engineer Newsletter are complementary r esources. This is despite the fact there is very little overlap in their contents.

Writing this book helped me kick off the newsletter because it was obvious there are plenty of timely software engineering topics to write about, which would make little sense to cover in a book with a longer lifespan than a weekly newsletter. The newsletter has helped me improve the book; I've learned lots about interesting trends and new tools that feel like they are here to stay for a decade or longer, such as AI coding tools, cloud development environments, and developer portals. These technologies are referenced in this book in much less detail than you will find in the newsletter.

I hope you discover useful ideas in this book, which serve you well for years to come.

A NOTE ON THE INDIAN EDITION

Before publishing this book, I had never been to India, and yet for years, I had known exactly what time it was in the country. I knew that 2:30 a.m. in Amsterdam meant it was 7 a.m. in India. Why? Because when I worked at Uber, I'd regularly be awoken at around 2:30 a.m. by a PagerDuty alert, indicating a payment method in India was experiencing an outage. This was because my team owned all of Uber's payment methods in India: PayTM, UPI, Jio Money, Airtel, Indian credit cards, and so on. These systems were built and maintained in the Netherlands.

I joined the ride-hailing app in 2016 as a software engineer on the Rider Payments team, which owned all digital payment methods at Uber. Back then, this included around 12 different payment methods, such as credit cards, Apple Pay, and all the methods used in India. Shortly after joining Uber, I became the team's engineering manager, meaning I had to understand more about how payments in India worked.

This gave me a front-row seat for the major events that shaped the payments space in India. In early November 2016, my team had just completed months of grueling work[2] to release Uber's new Rider app to the world. We launched the new app on 2 November 2016 and were nervous that we'd missed regressions. Sure enough, on 8 November, our oncall engineer was awoken in the middle of the night by an alert that signaled the cash payment functionality was having serious issues. The engineer looked through recent code changes and feature flag rollouts to uncover what could have caused cash payments to mostly stop working. They found nothing.

In a last-ditch effort, the engineer asked the Ops team in India if they could reproduce the issue. This is when they found the cause. The drop in cash was not a bug, but a feature: overnight, the Indian government had announced the demonetization of all ₹500 and ₹1,000 banknotes. As a result, cash transactions dropped close to zero for the next few days! Until then, I had no idea that political decisions could impact observability and oncall this much. We turned off all alerting for cash and other digital methods because our anomaly detection couldn't make sense of this major shift in how people paid; most notably, how PayTM became a preferred payment method across the business. I touch on oncall practices – some of which I learned from this incident – in Part V: "Reliable Software Systems."

I learned many interesting things about software engineering from my experience with payments in India. The country was, without doubt, the most complex payments market my team served, in terms of the complexity of code, the number of payment providers, and reliability challenges. So many personal learnings as a software professional came from work I did on these payment methods. For example, in 2017 my team added UPI support to Uber, as one of the first large apps to do this. At the time, I was skeptical about how well UPI would do, based on how cumbersome integration was, and how hard it was to debug payment failures. But UPI became one of the major success stories of democratizing payments within and outside of India. I touch more on the importance of understanding the company you work with in Part V: "Reliable Systems."

In 2019, college students in India were able to order ridiculous amounts of free food from UberEats, due to an issue with how my team integrated with PayTM[3]. The issue was that an endpoint we assumed was idempotent – basically, which we could invoke several times and expect one of two responses – quietly changed this idempotency and returned a new, unknown status code that we parsed as "success." I cover more about the dangers of working with an unknown state in Part II: "Coding."

And, of course, I worked a lot with engineers and engineering teams in India. After several years of owning India payment methods, I was able to make a business case for the India engineering team as a much better home for Uber's India payment methods, than Amsterdam was. Eventually, we moved the payment methods over. Working with engineering teams in different locations is always an interesting collaboration challenge, and this was no different. I cover this topic in Part III: "Collaboration and Teamwork."

[2]I am not kidding about the "grueling" part. We worked round the clock, sometimes 7 days a week for months. I write about my experience with this project at pragmaticurl.com/uber-rewrite

[3]https://pragmaticurl.com/ubereats-outage

I hope you'll enjoy the book and find the contents applicable to your day-to-day.

The Pragmatic Engineer Newsletter is available at a special reduced price for anyone in India. This newsletter is full of material for keeping up with the fast-moving tech industry. It is available to subscribe to for free. For paid subscriptions, I offer reduced pricing for readers in India, Pakistan, Bangladesh, Sri Lanka, Nepal, Bhutan or the Maldives:

pragmaticurl.com/india-newsletter

INTRODUCTION

This is the book I wish I could have read early in my career as a software developer; especially when I joined a larger tech company for a healthy pay rise, and found a very different engineering culture with surprisingly little guidance for navigating my new environment.

This book follows the structure of a "typical" career path for a software engineer, from starting out as a fresh-faced software developer, through being a role model senior/lead, all the way to the staff/principle/distinguished level. It summarizes what I've learned as a developer and how I've approached coaching engineers at different stages of their careers.

We cover "soft" skills which become increasingly important as your seniority increases, and the "hard" parts of the job, like software engineering concepts and approaches which help you grow professionally.

The names of levels and their expectations can – and do! – vary across companies. The higher "tier" a business is, the more tends to be expected of engineers, compared to lower tier places. For example, the "senior engineer" level has notoriously high expectations at Google (L5 level) and Meta (E5 level,) compared to lower-tier companies. If you work at a higher-tier business, it may be useful to read the chapters about higher levels, and not only the level you're currently interested in.

Naming and levels vary, but the principles of what makes a great engineer who is impactful at the individual, team, and organizational levels, are remarkably constant. No matter where you are in your career, I hope this book provides a fresh perspective and new ideas on how to grow as an engineer.

How to read this book

It is composed of six standalone parts, each made up of several chapters:

- Part I: Developer Career Fundamentals
- Part II: The Competent Software Developer
- Part III: The Well-Rounded Senior Engineer
- Part IV: The Pragmatic Tech Lead
- Part V: Role Model Staff and Principal Engineers
- Part VI: Conclusion

Part I and Part VI apply to all engineering levels, from entry-level software developer, to principal-and-above engineer. Part II, Part III, Part IV, and Part V cover increasingly senior engineering levels and group together topics in chapters, such as "Software Engineering," "Collaboration," "Getting Things Done," etc.

This is a reference book you can return to as you grow in your career. I suggest focusing on topics you struggle with, or the career level you are aiming for. Keep in mind that expectations can vary greatly between companies.

In this book, I've aligned the topics and leveling definitions to expectations at Big Tech and scaleups. However, there are topics that are also useful at lower career levels which we dive deeper into, later in the book. For example, in Part V: "Reliable Software Systems," we cover logging, monitoring, and oncall in-depth, but it's useful – and often necessary! – to know about practices below the staff engineer level. I suggest using the table of contents by topic, as well as by level when deciding which chapters to prioritize.

And now, let's jump in...

Part I

Developer Career Fundamentals

For the first few years of my developer career, I didn't care too much about career-related stuff. I assumed that if I worked hard and delivered good work, the rewards would follow. At developer agencies, promotions were rare and career development was much more limited, but I didn't feel like I missed out on anything while my title and level stayed the same at the first few places I worked.

It was when I moved to larger companies like JP Morgan and Microsoft that I noticed it's not always those who work hardest or deliver the highest quality work, who are awarded the biggest bonuses and win prized promotions. When I became an engineering manager at Uber, I had a team of engineers who required regular performance feedback and support in their professional growth, such as getting promoted to the next level.

This chapter summarizes observations of how different companies function, and career advice I gave to engineers on my team.

Something I wish I'd had a better understanding of much sooner is the different types of company you can work at as a developer – from well-known Big Tech giants, through startups, all the way to more traditional businesses, consultancies, and academia. Crucially, it can get increasingly hard to transition between different categories of workplace – which may be an unwelcome surprise a decade into working in one segment.

The other thing I missed out on was how to own your career. It was only when I became a manager that I realized what a difference it makes when a developer takes ownership of their career path – which also greatly helps their manager to advocate for them.

Most engineers I met assumed their manager would take care of most things career-related, and that glowing performance reviews and promotions would materialize as if from thin air. Perhaps this may conceivably happen at some small companies and startups, but at larger tech companies additional work is needed for career recognition. In most cases, it's not much additional work; it's just that many engineers don't know which additional activities to do.

The observations, concepts, and approaches in this chapter are applicable to all seniority levels, from entry-level engineer, all the way to staff-and-above.

CAREER PATHS

When it comes to careers, everyone's path is different. Some career elements are easy to be specific about, like where you work, your job title, and total compensation. However, many other important things are harder to measure, such as what working with colleagues is like, opportunities for professional growth, and work/life balance.

Career paths are diverse, and there's no simple way to define what a "good" career path looks like, as this varies from person to person. The best you can do is figure out which career path is interesting and achievable for you.

The routes people take into software engineering also vary; there are more common ways, like graduating from university with a computer science-related degree, and there are also self-taught engineers and career switchers. I worked with a colleague who'd been a chemical engineer for twenty years before teaching themself to code and becoming a developer.

In this chapter, we cover career-related topics:

1. Types of tech companies
2. Typical software engineering career paths
3. Compensation, and company "tiers"
4. Cost centers, profit centers
5. Alternative ways to think about career progress

1. Types of Companies

There's no way to categorize companies definitively, but they do share some common characteristics from the software engineer's point of view. Common types of company are:

Big Tech

Large, publicly traded tech businesses like Apple, Google, Microsoft, and Amazon. These typically employ tens of thousands of software engineers, and their market caps are in the billions of dollars.

Big Tech engineering jobs are usually the most sought-after, due to their top-of-the-market compensation, career growth opportunities beyond staff engineer level, and the chance to do work that impacts hundreds of millions of customers. There could also be the opportunity to work with best-in-class coworkers from across the industry.

Medium to large tech companies

Tech-first companies with software engineering at the core of their business. These companies are smaller than Big Tech and may employ hundreds or thousands of software engineers. Examples include Atlassian, Dropbox, Shopify, Snap, and Uber.

These companies tend to offer similar compensation to Big Tech, and career paths beyond staff engineer level. The user bases are usually somewhat smaller, but engineers' work can still impact tens of millions of customers.

Scaleups

Venture-funded, later-stage companies with product-market fits, which are investing in growth. These businesses may be making a loss on purpose, in order to invest in growing market share. Examples include Airtable, Klarna, and Notion.

These places usually move fast under high pressure to grow the business in order to justify a valuation, capitalize on future funding rounds, or prepare to go public.

A subset of scaleups is "unicorns:" businesses with a private valuation of $1B or more. In the 2010s, there were relatively few unicorns, and being one signaled a company might be the next big thing. Today, unicorns are much more common, so being one is less of a differentiator.

Startups

Venture-funded companies which have raised smaller rounds of funding, and are aiming for a product-market fit. This involves building a product that attracts customer demand.

Startups are inherently risky; they often lack meaningful revenue and are dependent on raising new rounds of funding to operate – for which a product-market fit is needed.

An example of a successful startup that graduated to a scaleup was Airbnb in 2011. Founded in 2008, the company won seed investment from Y Combinator. By 2010, Airbnb's product had gained traction, and it raised $7.2M Series A. In 2011, the company raised a $112M Series B, as investors saw its potential.

An example of a startup that didn't make it past the startup stage is Secret, an anonymous sharing app. Founded in 2013, Secret allowed users to share their secrets, incognito. The company enjoyed good traction and raised $35M in funding over two years. However, in 2015, it shut down and part of its funds were returned to investors.

Startups tend to offer the most freedom to software engineers, but also the least stability. These companies can also be demanding in work-life balance terms, as their existence depends on achieving a product-market fit before the money runs out. Meanwhile, founders greatly influence the environment of a startup. Some put a "work hard, play hard" culture in place, while others focus on a sustainable working culture. Startups offer the most variety of work, labor force, and growth opportunities.

Startups which offer employees equity are high-risk/high-reward places. If the business thrives and eventually exits via a public offering or acquisition, early employees with significant quantities of stock do very well financially. This happened at Airbnb when the company went public with a market cap of $86B in 2020, and at design collaboration tool Figma, which Adobe acquired for $20B in 2022.

Traditional, non-tech companies with tech divisions

These places have a core business with little to do with technology, which is just another division. Some are more than 50 years old and were founded before the software development era. Others are in sectors where technology is not a main source of value.

Examples of such companies include IKEA (home furnishing,) JPMorgan Chase & Co (financial services,) Pfizer (pharmaceutical,) Toyota (automotive,) and Walmart (retail.)

Many such companies have embarked on digitalization and aim to make software development more strategic to their businesses. However, the reality is that tech is more of a cost center than a profit center at these places, and these places tend to offer lower compensation than Big Tech

and many scaleups.

On the other hand, traditional companies tend to offer more job stability and a better work-life balance than most tech-first companies. The downside for software engineers is usually fewer career options than in Big Tech and at scaleups, and career paths above staff engineer are also rare.

Traditional but technology-heavy companies

An interesting subset of more traditional companies are those where technology is central to their offering in the form of hardware, software services, or both. These companies were often standout successes in their early days, and are now mature, reliable, and profitable businesses. However, with their age and slowdown in growth comes a more rigid organizational structure that is different from younger technology companies.

Examples of such companies include Broadcom, Cisco, Intel, Nokia, Ericsson, Mercedes-Benz and Saab. Hardware-heavy businesses, as well as automotive ones, can frequently be in this category.

These companies are commonly seen as less desirable to work at than Big Tech or scaleups. Compensation-wise, they are almost always below Big Tech and usually don't adopt new ways of working as quickly as younger companies do.

At the same time, these companies do offer complex engineering challenges that can be very satisfying to work on as an engineer, and the impact of your work can be at a scale that's more typical of Big Tech. They are also usually very stable places, and tend to offer a more predictable work-life balance than Big Tech or scaleups. Also, the tenure of software engineers at such companies can also be surprisingly lengthy, which contributes to predictability and stability that's much rarer at younger companies.

Small, non-venture funded companies

Bootstrapped companies, family businesses, and lifestyle businesses (which exist for the benefit of the founder's lifestyle,) are all examples of smaller companies without venture funding. This means two things:

1. No investor pressure to grow at all costs
2. Profitability is required, or the business will fail

These characteristics mean such small companies are rarely high-growth and are conservative in hiring and their business approach. However, they can be friendly, stable places to work, thanks to a comfortable pace of work, profitability, and because many people choose to stay longer at stable companies than at more hectic places.

Public sector

There is constant demand for governments to invest in software development, and they do.

The upside of public sector jobs is stability, and that compensation is usually clearly communicated and follows a formula. Many positions also have healthy perks in terms of time off and benefits.

Downsides may include a slow, bureaucratic approach, and the need to support legacy systems which are hard to change. In some countries, it can also be harder to move from a government job to the private sector.

An example of a government organization with a good reputation is the UK's Digital Service division, which builds and maintains many state websites. It is exemplary in how it works; for example, by publishing much of its work on GitHub (at https://github.com/alphagov.) Another public sector organization with a good engineering culture is the British Broadcasting Corporation (BBC.)

Nonprofits

These exist to serve a public or social cause. Examples include Code.org, Khan Academy, and the Wikimedia Foundation.

Non-profits typically offer less compensation than venture-funded companies, but in return they have a different mission than generating returns for investors and profits for owners. Working environments vary; some are excellent places for technologists to work, but tech is usually more of a cost center.

Consultancies, outsourcing companies and developer agencies

So far, we've covered companies that build products and services, for which they employ software engineers. However, there is considerable demand for "renting" software engineering expertise

via an agency or outsourcing provider.

Outsourcing companies provide as many software engineers as a customer needs, and the customer decides where to allocate the engineering talent within its business. Meanwhile, a consultancy makes contracts with customers to build complex projects end-to-end, by providing software engineers who do this work. The consultancy is typically responsible for building and shipping the whole project.

Examples of consultancies and outsourcing companies include Accenture, Capgemini, EPAM, Infosys, Thoughtworks, and Wipro.

Developer agencies are usually small to midsize companies that take on smaller consulting projects, like building websites, apps, and similar projects for clients. They can also handle service maintenance for customers. Consultant engineers often bill customers at a daily or hourly rate, while being full-time employees of the developer agency.

The upside for software engineers of consultancies, outsourcing companies, and developer agencies, is that they're usually the easiest places to get hired, especially if you're less experienced. This is because these companies often have high demand for talent, and offer less compensation than their customers in the other categories.

Other upsides are that training for less experienced engineers is often available, there's opportunities to work on a variety of projects, and to get a peek into many different workplaces as a contractor.

There are downsides to working at consultancies. The most common:

- Career development-wise, these companies usually don't offer paths to above staff engineer-level, which is one step above senior engineer
- The scope of work is limited to what the customer sets. Consultancies are generally hired for projects that a customer considers to be outside its core competency.
- Not much focus on good software engineering practices. Clients pay for short-term results, not for a developer to work on long-term things like reducing tech debt.
- It might be hard to switch to product-focused companies later. Companies that build products like Big Tech, startups, and scaleups tend to have very different cultures where maintainability is important, as is taking the initiative. Working at a consultancy for too long can make the switch to these places harder.

Academia and research labs

These institutions usually are part of, or work closely with, universities, and work on long-term research projects. Some focus on applied research, while others do basic research.

An upside of working in research labs is applying your skills to less explored fields, and the stability of being in an environment with fewer to no commercial pressures.

Which type of company best fits your career goals?

As we see, there are many types of companies and organizations to consider as a software engineer. So which one fits you best?

There's unlikely to be opportunities simultaneously at all ten types of place listed above. So narrow the list down to realistic options, based on your circumstances. It's helpful to talk with friends, family, and people in your network who are engineers, if possible. Get their opinion on whether they like their workplace, and what their job is really like.

Don't forget, there can be huge differences between companies in the same category, and teams differ significantly in the same workplace. Someone working in a traditional business's tech division on a fantastic team likely has a better time than someone in a struggling team at a Big Tech giant.

2. Typical Software Engineering Career Paths

Career paths for software engineers are pretty simple within a *company*. The two most common are the single-track and the dual-track career paths.

Single-track career path

The single-track career path of an individual contributor (IC) and a manager typically looks like this:

Level	Individual contributor	Manager
1	Software Engineer	
2	Senior Engineer	
3	Staff/Principal Engineer	Manager
4		Director
5		VP of Engineering
6		CTO

A typical single-track career path. Compensation rises by level, as do expectations

At smaller and non-tech first companies, there's a de facto career ceiling for software engineers at level 3 – the staff/principal level –, beyond which growth is only possible by switching to the manager track.

One downside of switching paths is that plenty of engineers come to believe management isn't for them, and so they quit to work elsewhere, meaning an employer loses some of their best engineers who become managers, or who exit the business entirely because they dislike managing.

Dual-track career path

Companies where engineering grows to above 30-50 people, or which are more forward-thinking, often have a dual-path career ladder, to avoid engineers having to choose between being stuck at a level with similar compensation to a line manager or becoming managers themselves. This is what the dual-path approach typically looks like:

Level	Individual contributor ladder	Manager ladder
1	Software engineer	
2	Senior engineer	
3	Staff engineer	Manager
4	Senior staff engineer	Director
5	Principal engineer	Senior director
6	Distinguished engineer	VP of engineering
7	Fellow	Senior VP of engineering
8		CTO

A typical dual-path career ladder: Once again, compensation rises by level, as do expectations

At companies with dual-path career progression, there are a few different approaches:

1. Individual contributor. Progress from software engineer, through senior, and to increasingly challenging IC levels.
2. Engineering manager. Change to an engineering manager from senior or staff engineer, then progress along the management track. With some luck, get to director-and-above positions.
3. Switch between the IC and manager tracks. Go back to being an engineer after a manager role, and perhaps repeat in future. This approach is more common than you might expect at such companies

All career paths are unique

In reality, many people switch jobs pretty frequently in tech. The majority of software engineers I've known do switch jobs every few years, which changes their career trajectory. This creates varied career paths, which you can see in the LinkedIn profiles of experienced former software engineers. Here's a few examples.

The 20-year career path of Tanya Reilly[1], author of the excellent book, "The Staff Engineer's Path," is pretty linear and has stayed on the software engineer path:

- systems administrator (Fujitsu, Eircom)
- → software engineer (Google)
- → senior software engineer (Google)
- → staff systems engineer (Google)
- → principal software engineer (Squarespace)
- → senior principal software engineer (Squarespace)

The 20-year career path of Nicky Wrightson[2] – a head of engineering a the time of publishing –, stayed on the software engineering path for a long time, before moving into leadership:

- software developer (consultancy)
- → specialist consultant and developer (telecom companies)
- → developer (BNP Paribas, JP Morgan, Morgan Stanley)
- → six-month sabbatical
- → principal engineer (Financial Times, River Island, Skyscanner)
- → fractional CTO (venture builder firm Blenheim Chalcot)
- → Head of Engineering (topi)

The career path of Mark Tsimelzon[3] – director of engineering at WhatsApp at the time of publishing – has varied over three decades, from starting as a software engineer, to alternating between being a founder and building stuff, and leadership positions:

- software engineer
- → engineering manager
- → founder (startup later acquired by Akamai)
- → product manager (Akamai)
- → founder (at a startup later acquired by Sybase)
- → director of engineering (Yahoo)
- → entrepreneur in residence (venture capital firm)
- → VP of engineering (startup acquired by Yahoo)
- → senior director of engineering (Yahoo)

---

[1] https://www.linkedin.com/in/tanyareilly/

[2] https://www.linkedin.com/in/nickywrightson/

[3] https://www.linkedin.com/in/marktsimelzon/

- → VP of engineering (Syapse)
- → chief engineering officer (Babylon Health)
- → director of engineering (Meta)

Common career paths

The software engineering industry is dynamic and ever-changing, so it should be little surprise that software engineers have lots of opportunities to shape their careers by taking opportunities that come along. I've observed at least a dozen "common" career paths:

1. **The lifelong software engineer**. An engineer who stays a software engineer, becoming increasingly senior (e.g. senior engineer, staff, principal engineer,) and moving between companies. They often move stacks and broaden their skillset with each new position.
2. **The software engineer turned specialist**. A developer who comes to specialize in a domain, like native mobile or backend, and stays for an extended time.
3. **Generalist/specialist pendulum**. An engineer who specializes in a technology, and then spends time in a more generalist position. Rinse and repeat.
4. **Software engineer specializing in a niche field**. For example, a software engineer who transitions to a site reliability engineer, or a data engineer, where they do some coding, but the work mostly doesn't resemble software engineering.
5. **Software engineer turned contractor/freelancer**. Having reached senior engineer status, this person becomes a contractor or freelancer, often earning more and worrying less about internal politics and career development.
6. **Software engineer turned tech lead**. A developer who starts by leading a team, though not necessarily doing management tasks. Even when they switch jobs, they eventually return to the tech lead position.
7. **Software engineer turned engineering manager**. A developer who becomes an engineering manager, and then progresses on that career path.
8. **Software engineer turned founder/business owner**. Following a career as a software engineer, this person starts or co-founds a business.
9. **Software engineer turned non-software engineer**. A person who moves into another tech field like developer relations (DevRel,) product management, technical program management (TPM,) tech recruitment, or other. Their experience as a software engineer is relevant, and they get to explore fields they've developed an interest in.
10. **Software engineer/manager pendulum**. A software engineer who becomes an engineering manager, and then returns to being an engineer, often repeating this switch a few times. CTO Charity Majors writes about this increasingly common path in The

Engineer/Manager Pendulum[4].

11. **Combination of some of the above**. For example, a software engineer turned engineering manager, turned product manager, turned founder, or any other combination.
12. **A non-linear career path**. For example, a software engineer who becomes an engineering manager, then takes a long career break to have a family, or pursue a different career. They return to the field as a director of engineering. Non-linear career paths develop in ways unique to each person.

In this book, we explore a more typical career path within a company, from entry-level engineer to staff+ engineer. However, while such a career path is typical for an IC progressing within a company, it's not necessarily common for software engineers. By detailing the sheer variety of career paths from starting as a software engineer, I hope to show there's not one "good" career path. Opportunities and preferences vary by person, and you should feel empowered to venture into less conventional areas.

3. Compensation and "Tiers" of Companies

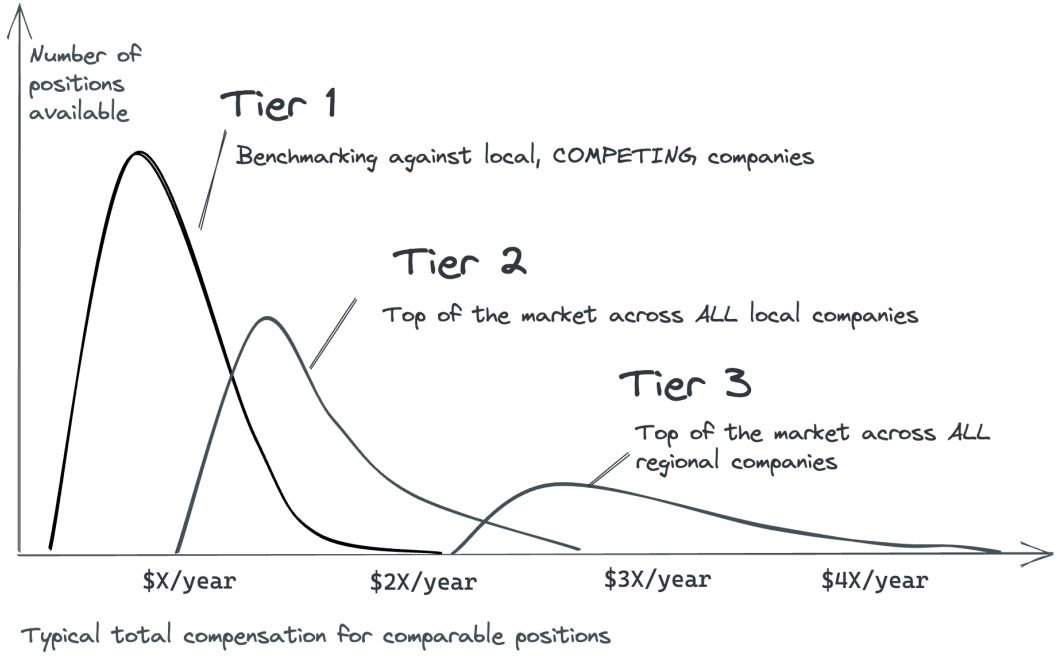
It is tricky to quantify every aspect of a job, its professional challenges, flexibility, and work/life balance.

One thing that is easier to quantify is total compensation – with the caveat that equity packages in private companies can be tricky to quantify. As a hiring manager, I've observed what seemed like a trimodal distribution of compensation packages, and also confirmed this distribution by analyzing thousands of self-reported data points submitted to TechPays.com – a site I built prior to the publication of this book. Here's the distribution:

The same role and title can vary in compensation by a factor of 2-4 times at different types of companies, the data reveals. A senior engineer at Google is likely to make at least double – and potentially even quadruple – the total compensation of a senior engineer at a non-tech company, or a small family-run business. Total compensation means the combination of three values:

- Base salary
- Cash bonus

[4]https://charity.wtf/2017/05/11/the-engineer-manager-pendulum/



The trimodal nature of software engineering salaries: Why a comparable position can (and does) pay very differently. Tier 1: local companies; Tier 2: top of the local market; Tier 3: top of the regional market

- Equity: at publicly traded companies, this is liquid, and at privately owned startups and scaleups it is illiquid.

Let's talk about the three tiers of the tech market as they relate to compensation.

Tier 1: local market

Companies that benchmark against the local market. Companies in this group are typically:

- Local startups
- Small, non-venture funded companies
- Traditional non-tech companies with tech divisions
- Public sector
- Nonprofits

- Consultancies, outsourcing companies, and developer agencies
- Academia and research labs

Tier 2: top of the local market

Companies that aim to pay at the top of the local market to attract and retain standout local talent. Typical companies in this group:

- Some medium-sized tech companies, especially ones that optimize compensation for the local market
- Some scaleups
- Some startups, usually those with healthy funding and a regional focus
- Some traditional companies with tech divisions, especially those doubling down on tech investment

Tier 3: top of the regional/international market

Companies that aim to pay the best across the region, and compete with peer tier 3 companies, not local rivals. Typical companies in this group:

- Big Tech
- Most medium-sized tech companies
- Well-funded scaleups which compete for talent with Big Tech and midsize tech companies
- Startups with strong funding that hire from the above groups

Compensation at tier 3 companies tends to have three components:

- Base salary
- Equity, issued annually
- Cash bonus, issued annually

When publicly traded companies issue stock, it can be sold upon vesting as part of the total compensation package. This is how staff-level software engineers at Big Tech and similarly large companies earn more in stock compensation annually than their base salary.

Many startups and scaleups also issue equity for software engineers. However, at private companies, this equity is not liquid and so any stock increase is "paper gains" only, until the company has an exit like a public offering or is acquired. These exits are how early employees with significant stock holdings make small fortunes, as happened to early Uber employees. Of course,

privately traded companies are riskier because many never achieve an exit. This happened to early Foursquare employees in 2023 when their stock grants simply expired[5] and "vanished," 14 years after the company was founded.

Contractors and freelancers

So far, we've covered fulltime total compensation. However, we need to look into compensation for contractors and freelancers, who often do similar work to permanent fulltime employees but are paid differently.

They bill the company an agreed rate, usually hourly or daily. From the Human Resources point of view, they're not employees and have a business-to-business contract to provide software engineering services to a client. The terminology varies by country; the US and UK use contractors, while in many European countries they're called freelancers. This book uses "contractor."

As a contractor, a major contrast to being fulltime is that compensation can be significantly higher. The rates senior-and-above software engineers can charge as contractors, almost always place them above Tier 2 compensation packages. Meanwhile, some high-end contractors can earn the equivalent of Tier 3 compensation packages.

In some countries – especially in Europe – fulltime employment income is heavily taxed, while income from contracting may be less so.

From an employer's point of view, the biggest difference between contractors and fulltime workers is flexibility: they can recruit contractors quickly and terminate their contracts equally rapidly. Also, there's no need to worry about career progression, training, or severance terms. Vacation is also usually outside the scope of such agreements; when a contractor takes a day off, they don't bill for that day.

Performance management and the career ladder are also different for contractors. There are no performance reviews or opportunities for promotion. As a result, there's no need for many of the activities full-time employees do for good performance management outcomes. Plenty of people choose contracting because they don't want more career progression based on internal company levels, and welcome the opportunity to focus more on their work, and less on performance management processes and office politics.

[5] https://www.theinformation.com/articles/the-private-tech-company-that-let-employee-stock-grants-evaporate

A downside for contractors is that their job is often less secure than an equivalent permanent role. Fulltime employees' jobs are protected by rules and regulations in most countries. But for contractors, the redundancy process is very simple by design; usually, the employer simply does not extend a fixed-term contract or else serves notice according to a contractual timeframe. Contractors are very easy to hire and fire, so tend to be the first team members who are let go when times get tough.

However, contractors tend to be comfortable with lower job stability in return for higher compensation. Many are senior-or-above engineers who are content to not climb the corporate career ladder.

Tradeoffs between tiers

How do company tiers, based on compensation philosophies, compare? It's tricky to be objective as each company is different, and each working environment has upsides and downsides. Below are some observations about tiers 1, 2, and 3 for full-time employees. Contractors are not covered in detail, as there are no assigned levels or promotions to aim for. However, great contractors tend to embody many traits we cover in the "Senior" and "Staff" sections of this book.

Area	**Tier 1 (local)**	**Tier 2 (top of local)**	**Tier 3 (top of regional)**
How hard to get a job	Easiest	More challenging	Very challenging
Performance expectations	Usually reasonable	Often demanding	Almost always demanding
Career paths as an individual contributor	Usually up to senior or staff	Sometimes beyond staff	Almost always beyond staff
Work/life balance	Can be a focus	Usually less of a focus	Usually less of a focus

Comparing the three tiers of companies

The highest-paying companies predictably tend to attract the most candidates for jobs, which means they can – and usually do – place the highest demands upon software engineers.

4. Cost Centers, Profit Centers

Many companies apply the concept of "profit centers" and "cost centers" to their business. Which one of these you work in can have implications for your career.

A profit center is a team or organization that directly generates revenue for the business. A classic example is the Ads organization at Google, which is directly responsible for generating the majority of Google's income. There are many teams that help with this effort; the Search team brings visitors to the site and therefore also contributes heavily, for example. But without the Ads team building tools for advertisers to spend their ad budgets, Google would make much, much less money.

A cost center refers to teams or organizations which do not directly generate revenue, but are still needed for the company to operate smoothly. A good example is an engineering team working on compliance, ensuring the company is GDPR-compliant in Europe. Their activities are required, but generate no revenue, and so are cost centers from a business point of view.

What are the implications of working at cost centers or profit centers at larger companies? Here are a few:

- **Promotions**: these are almost always easier in profit centers. It's easier to sell a promotion case by displaying the impact on revenue generation. An exception is at Big Tech, where for staff-and-above positions, solving organization-wide engineering challenges is an expectation on top of organization-wide business impact. Such requirements incentivize experienced engineers to work on platform teams in order to progress their careers.
- **Performance reviews and bonuses**: there's usually no difference in working at either type of organization between entry-level and up to senior engineering level. At higher levels, those in profit centers frequently get better "scores" and bonuses. This is because most businesses naturally tilt toward money-makers when all else is equal in engineers' contributions.
- **Internal transfers**: it's understandable if workers want to be in a profit center. However, this is frequently not the case; many engineers are drawn to complex and interesting work, and that mostly happens on "Big Bet" projects which are not (yet) profit centers. Conversely, profit centers are often among the more "boring" teams in an organization, so fewer people want to join. Imagine getting into Meta and choosing a team to work on; would you rather work on Ads infrastructure and increase ad revenue by 0.005%, or a new team building an exciting, innovative way to connect with friends?

- **Attrition**: cost centers almost always have higher turnover as more employees leave the company, or transfer to profit centers because career advancement in these teams can be easier.
- **Job security**: cost centers are a prime target for job cuts when a company needs to make cost savings.

So, how do you know if your team or organization is a profit or cost center? Here are some ways to find out:

- Does your team or organization report its revenue generated in every period? If so, you're likely in a profit center.
- How does your company make money, and which organizations bring in revenue? Does Sales get all the credit, or is it the front office if you work at a bank? Is Tech credited for generating revenue? Which teams within Tech get credit?
- Look at the org chart. How high up is technology represented in the organization? To where does engineering and product report? How many VPs are there in engineering, compared to marketing, finance, operations and other groups?
- Which teams do the CEO call "strategic" during all hands meetings, and credit for increasing revenue? Is your team or organization among them?
- Is your company publicly traded? If so, read quarterly reports for a sense of the corporate focus, which will likely be areas where profit centers are.

Software engineering can be a cost or profit center:

- In Big Tech, at midsize tech companies, and at startups and scaleups where tech is core to business activities, tech and software engineering are frequently seen as cost centers.
- At traditional companies and public sector employers, it's common to treat tech as a cost center that's a "means to an end."
- At consultancies and developer agencies, development is what the company provides as a service, meaning this activity is usually a profit center.

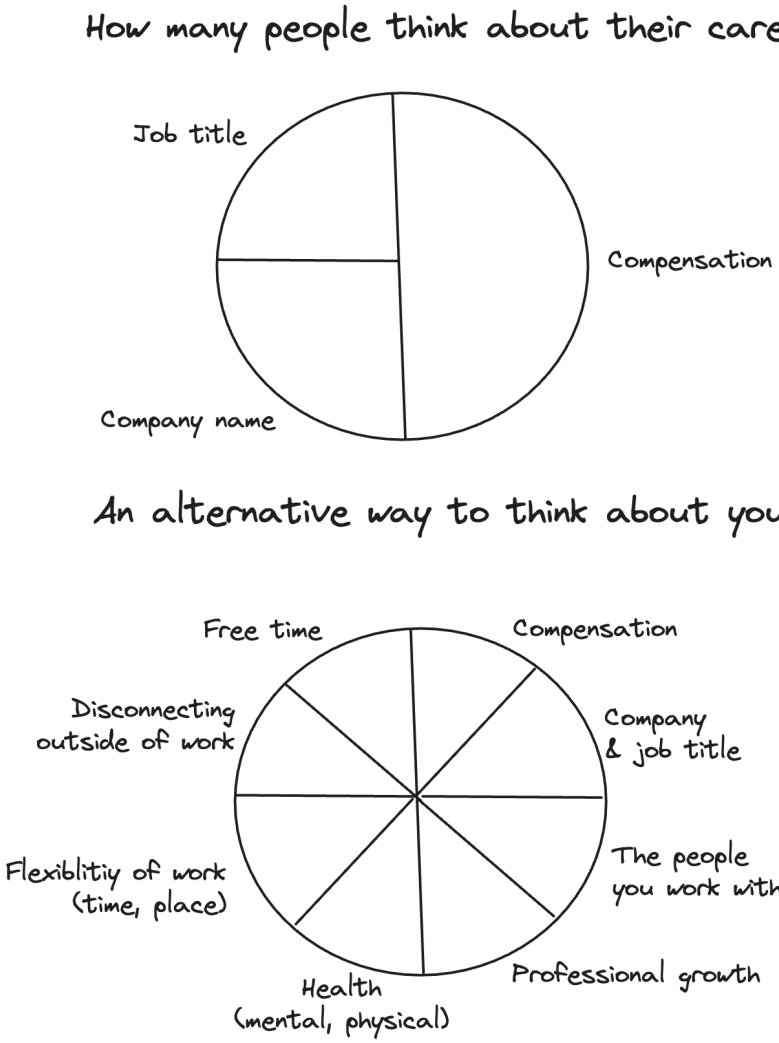
Working both in cost centers and profit centers gives you perspective. It's easy to feel superior to cost centers when you work in a profit center. However, effective companies need both types of teams and organizations, so it's a useful skill to know how to thrive in each.

5. Alternative ways to think about career progress

Believe it or not, there is much more that matters in your job and career, than titles and compensation. Your rank, a company's reputation, and pay are the easiest things to talk about because they're concrete, and compensation numbers offer an easy way to compare positions. Here are some other factors that contribute to how satisfying your job is:

- People whom you work with and team dynamics
- Your manager and your relationship with them
- Your position within the team and the company
- Company culture
- The mission of the company, and what it contributes to society
- Professional growth opportunities
- Your mental and physical health in this environment
- Flexibility. Can you work remote, or from home? If so, how often, and on what notice?
- Oncall. How demanding and stressful is it?
- Life beyond work: how easy is it to "leave work at work?"
- Personal motivations

Visualizing this:



An alternative way to think about where you are in your career

Weigh the areas in the charts above against the compensation packages of the positions you apply for. It's not uncommon for long-serving professionals to take a pay cut in order to get an "upgrade" in one or more of these areas. Find the balance which works for you in your job, and in the next one. You will have a much more satisfying career than people who optimize only

for easy-to-measure parts of jobs.