O'REILLY®

The Staff Engineer's Path

A GUIDE FOR INDIVIDUAL CONTRIBUTORS NAVIGATING GROWTH AND CHANGE

TANYA REILLY

Foreword by Camille Fournier, author of *The Manager's Path*

O'REILLY®

"The book I wish I'd had when I stepped up to principal engineer. If you are wondering what Staff+ means, and how to be successful in the role in your organization, Tanya has laid out the path, with lots of practical, insightful advice for you. This book will provide the tools to enable you to thrive in a tech track role, acting with influence and impact."

—Sarah Wells, independent consultant and author, former principal engineer at the *Financial Times*

The Staff Engineer's Path

A GUIDE FOR INDIVIDUAL CONTRIBUTORS NAVIGATING GROWTH AND CHANGE

For years, companies have rewarded their most effective engineers with management positions. But treating management as the default path for an engineer with leadership ability doesn't serve the industry well—or the engineer. The staff engineer's path allows engineers to contribute at a high level as role models, driving big projects, determining technical strategy, and raising everyone's skills.

This in-depth book shows you how to understand your role, manage your time, master strategic thinking, and set the standard for technical work. You'll read about how to be a leader without direct authority, how to plan ahead to make the right technical decisions, and how to make everyone around you better, while still growing as an expert in your domain.

By exploring the three pillars of a staff engineer's job, Tanya Reilly, a veteran of the staff engineer track, shows you how to:

- Take a broad, strategic view when thinking about your work
- Dive into practical tactics for making projects succeed
- Determine what "good engineering" means in your organization

Tanya Reilly has over twenty years of experience in software engineering, most recently working on architecture and technical strategy as a senior principal engineer at Squarespace. Previously she spent twelve years at Google, responsible for some of the largest distributed systems on the planet. She's an organizer and host of the LeadDev StaffPlus conference. Originally from Ireland, she now lives in Brooklyn.

BUSINESS / SOFTWARE ENGINEERING

US $45.99 CAN $57.99
ISBN: 978-1-098-11873-0
9 781098 118730 54599

Twitter: @oreillymedia
linkedin.com/company/oreilly-media
youtube.com/oreillymedia

Praise for *The Staff Engineer's Path*

The book I wish I'd had when I stepped up to principal engineer. If you are wondering what Staff+ means, and how to be successful in the role in your organization, Tanya has laid out the path, with lots of practical, insightful advice for you. This book will provide the tools to enable you to thrive in a tech track role, acting with influence and impact.

—*Sarah Wells, independent consultant and author, former principal engineer at the* Financial Times

This book feels like the missing manual for my whole career. It's amazingly reassuring to see the ambiguity of the role laid out in print, along with great specific guidance on time management, consensus building, etc. I'm going to cite this a lot.

—*Titus Winters, principal engineer, Google, and coauthor of* Software Engineering at Google

Tanya is the perfect author for this exceptional guide to navigating the murky role of staff-plus engineering. Her deep, direct experience comes through in every section and taught me a great deal.

—*Will Larson, CTO, Calm, and author of* Staff Engineer

The job of senior leadership as an individual contributor has long been ambiguous and difficult to define, and this book is a much-needed guide on being successful in a relatively new role to our industry. Tanya does an excellent job bringing large-company perspective and scaling company challenges for a rounded view on how to be a successful staff engineer.

—*Silvia Botros, principal engineer and coauthor of* High Performance MySQL*, 4th edition*

When you reach near the top of the individual-contributor scale, you're given a metaphorical compass and a destination. How you get there is your problem. How you lead there is everybody's problem. Tanya offers a solid framework, a mapping approach, to help you lead from "here" to "there." This book offers a solid anchor for those new to the upper levels of individual contributors, and new perspectives for those with more experience. Staff engineer, know thyself.

—*Izar Tarandach, principal security architect and coauthor of* Threat Modeling

Tanya Reilly captures with eerie accuracy the sinking feeling I experienced when I first became the "someone" in "someone should do something." This book is a detailed exploration of what that actually means for folks at the staff engineer level.

—*Niall Richard Murphy, founder, CEO, and coauthor of* Reliable Machine Learning *and* Site Reliability Engineering

In *The Staff Engineer's Path*, Tanya Reilly has brought desperately needed clarity to the ambiguous and often misunderstood question of how to be a senior technical leader without direct reports. Every page is chock full of valuable insights and actionable advice for navigating your role, your org, and carving out your career path—all delivered in Tanya's trademark witty, insightful, and down-to-earth style. This book is a masterpiece.

—*Katie Sylor-Miller, senior staff frontend architect, Etsy*

If you're a senior engineer wondering what the next level is—a staff-level engineer or a manager of staff engineers—this book is for you. It covers so many of the things no one tells you about this role—things that take long years, even with great mentors, to discover on your own. It offers observations, mental models, and firsthand experiences about the staff engineer role in a more distilled way than any other book has covered before.

—*Gergely Orosz, author of* The Pragmatic Engineer

The Staff Engineer's Path

A Guide for Individual Contributors Navigating Growth and Change

Tanya Reilly

Beijing • Boston • Farnham • Sebastopol • Tokyo O'REILLY®

The Staff Engineer's Path
by Tanya Reilly

Copyright © 2022 Tanya Reilly. All rights reserved.

Printed in the United States of America.

Published by O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472.

O'Reilly books may be purchased for educational, business, or sales promotional use. Online editions are also available for most titles (*http://oreilly.com*). For more information, contact our corporate/institutional sales department: 800-998-9938 or *corporate@oreilly.com*.

Acquisitions Editor: Melissa Duffield
Development Editor: Sarah Grey
Production Editor: Elizabeth Faerm
Copyeditor: Josh Olejarz
Proofreader: Liz Wheeler
Indexer: Sue Klefstad
Interior Designer: Monica Kamsvaag
Cover Designer: Susan Thompson
Cover Art Creator: Susan Thompson
Illustrator: Kate Dullea

September 2022: First Edition

Revision History for the First Edition

2022-09-20 : First Release
2023-01-06 : Second Release

See *http://oreilly.com/catalog/errata.csp?isbn=9781098118730* for release details.

The O'Reilly logo is a registered trademark of O'Reilly Media, Inc. *The Staff Engineer's Path*, the cover image, and related trade dress are trademarks of O'Reilly Media, Inc.

The views expressed in this work are those of the author, and do not represent the publisher's views. While the publisher and the author have used good faith efforts to ensure that the information and instructions contained in this work are accurate, the publisher and the author disclaim all responsibility for errors or omissions, including without limitation responsibility for damages resulting from the use of or reliance on this work. Use of the information and instructions contained in this work is at your own risk. If any code samples or other technology this work contains or describes is subject to open source licenses or the intellectual property rights of others, it is your responsibility to ensure that your use thereof complies with such licenses and/or rights.

978-1-098-11873-0

[LSI]

Contents

Foreword

When I wrote *The Manager's Path* in 2016, I had many goals. I wanted to share lessons I had learned growing up as a manager. I wanted to show those who were interested in becoming managers what the job would be like. And I wanted to force a reckoning across the industry that we needed to expect more from our managers, and that the managers we were currently promoting often did not have the right balanced focus of people, process, product, and technical skills to do the job well. In short, I wanted to correct what I saw as a cultural failing in tech: to both take management seriously as a critical role and to discourage it from being the default path for ambitious engineers who want to grow their careers.

I would say that I partially succeeded. Every time someone tells me they read my book and decided not to become a manager, I do a little victory dance. From that perspective, at least some people read my book and realize that this path isn't for them. Unfortunately, the alternative path of career growth for the individual contributor, the staff+ engineering path, has lacked a similar guidebook. This lack has led to many choosing to follow the management path despite knowing they would rather not have the responsibility for larger and larger groups of people, because they cannot see another way forward. This is a great frustration for engineers and managers alike: most managers want to have more strong staff+ engineers in their organizations but don't know how to cultivate them, and many engineers want to stay on that path but see no realistic options beyond going into management.

One of the core challenges of the staff+ engineering path is the unspoken expectation that part of being qualified to be on that path is figuring out how to climb it without much in the way of directions. If you were destined to be a staff+ engineer, conventional wisdom argues, you would figure out how to get there yourself. Needless to say, this is a frustrating and bias-ridden approach to career development. As more and more companies realize the need for staff+ engineers, we as an industry cannot afford to maintain a mysticism about the staff+ engineering career path that ignores the underlying skills that lead to successful technical leaders.

With this in mind, you can imagine how thrilled I felt when Tanya Reilly came up with a proposal for a book about career growth as a staff engineer, filling in the missing half of the career ladder that my book left unexplored. I know Tanya from her writing and speaking about technical leadership, and it is obvious that she wants to correct the cultural failings of the tech industry's approach to staff+ engineering in the same way that I wanted to correct the cultural failing in the tech industry's approach to management. Namely, Tanya wants to address the overfocus on coding and technical contributions and the lack of clarity around the skills that allow strong engineers to become successful multipliers without needing to manage people.

In this book, Tanya has stepped up to the task of articulating these underlying skills that are so crucial for successful staff+ engineers. She provides a framework that shows how, using the pillars of big-picture thinking, execution, and leveling up others, you can build impact that goes beyond your individual hands-on contributions.

Reflecting the multifaceted nature of the staff+ engineering path, Tanya does not try to dictate the precise mix of skills needed at each level above senior engineer. Instead, she wisely focuses on how to build out these pillars from wherever you are today. From developing technical strategy to leading big projects successfully and going from mentor to organizational catalyst, this book takes you through these critical pillars and shows how to increase your impact on the success of your company beyond writing code.

There's only one person in the driver's seat for your career, and that person is you. Figuring out your career path is one of the great opportunities and challenges of your life, and the earlier you accept that it's up to you (plus a heaping dose of luck), the better set you are to navigate the working world. This guidebook shows you the skills you'll need on the staff+ path, and it's an essential addition to every engineer's library.

—Camille Fournier
Author, The Manager's Path
Editor, 97 Things Every Engineering Manager Should Know
Managing Director, JP Morgan Chase
Board Member, ACM Queue
New York, NY, September 2022

Introduction

Where do you see yourself in five years? The classic interview question is the adult equivalent of "What do you want to be when you grow up?": it has some socially acceptable answers and a long enough time horizon that you don't need to commit.[1] But if you're a senior software engineer looking to keep growing in your career, the question becomes very real.[2] Where *do* you see yourself going?

Two Paths

You may find yourself at a fork in the road (Figure P-1), two distinct paths stretching ahead. On one, you take on direct reports and become a manager. On the other, you become a technical leader without reports, a role often called *staff engineer*. If you really could see five years ahead on both of these paths, you'd find that they have a lot in common: they lead to many of the same places, and the further you travel, the more you'll need many of the same skills. But, at the start, they look quite different.

1 Although you're not supposed to reply "a zookeeper who is also an astronaut" to the interview question. Adult life is very limiting.

2 For the sake of brevity, I'm going to say "software engineer" throughout this book; however, if you're a systems engineer, data scientist, or any other practitioner of tech, I think you'll find it relevant, too. All are welcome here!

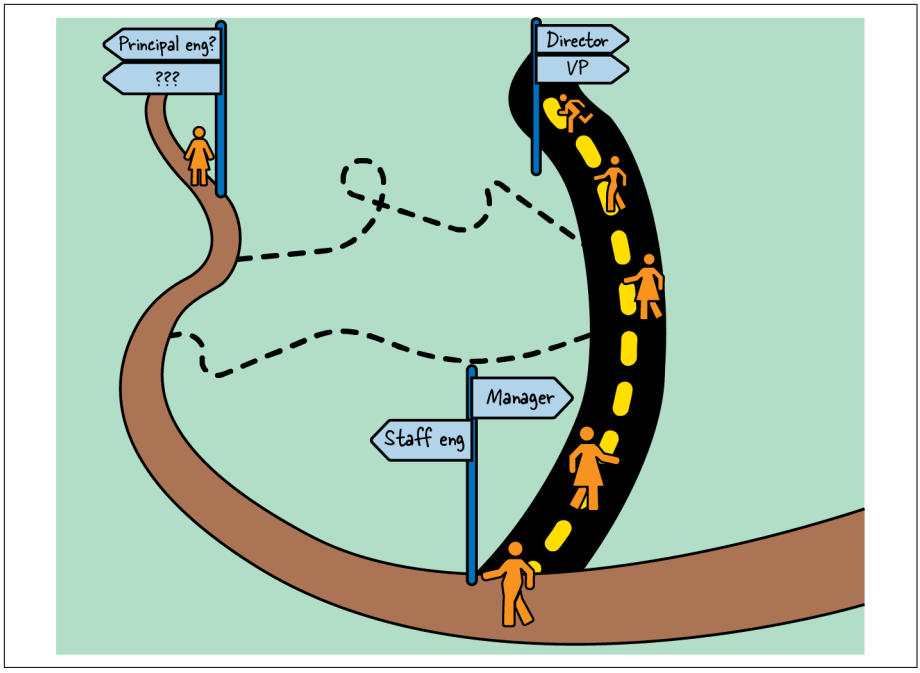

Figure P-1. A fork in the road.

The manager's path is clear and well traveled. Becoming a manager is a common, and perhaps default, career step for anyone who can communicate clearly, stay calm during a crisis, and help their colleagues do better work. Most likely, you know people who have chosen this path. You've probably had managers before, and perhaps you have opinions about what they did right or wrong. Management is a well-studied discipline, too. The words *promotion* and *leadership* are often assumed to mean "becoming someone's boss," and airport bookshops are full of advice on how to do the job well. So, if you set off down the management path, it won't be an *easy* road, but you'll at least have some idea of what your journey will be like.

The staff engineer's path is a little less defined. While many companies now allow engineers to keep growing in seniority without taking on reports, this "technical track" is still muddy and poorly signposted. Engineers considering this path may have never worked with a staff engineer before, or might have seen such a narrow set of personalities in the role that it seems like unattainable wizardry. (It's not. It's all learnable.) The expectations of the job vary across

companies, and, even within a company, the criteria for hiring or promoting staff engineers can be vague and not always actionable.

Often the job doesn't become clearer once you're in it. Over the last few years, I've spoken with staff engineers across many companies who weren't quite sure what was expected of them, as well as engineering managers who didn't know how to work with their staff engineer reports and peers.[3] All of this ambiguity can be a source of stress. If your job's not defined, how can you know whether you're doing it well? Or doing it at all?

Even when expectations are clear, the road to achieving them might not be. As a new staff engineer, you might have heard that you're expected to be a technical leader, make good business decisions, and influence without authority—but *how*? Where do you start?

The Pillars of Staff Engineering

I understand that feeling. Through 20 years in the industry, I've stayed on the staff engineer's path, and I'm now a senior principal engineer, parallel to a senior director on my company's career ladder. While I've considered the manager's path many times, I've always concluded that the "technical track" work is what gives me energy and makes me want to come to work in the morning. I want to have time to dig into new technologies, deeply understand architectures, and learn new technical domains. You get better at whatever you spend time on, and I've consistently wanted to keep getting better at technical things.[4]

Earlier in my career, though, I struggled to make sense of this path. As a midlevel engineer, I didn't understand why we had levels above "senior"—what did those people *do* all day? I certainly couldn't see a path to those roles from where I was. Later, as a new staff engineer, I discovered unspoken expectations and missing skills I didn't know how to *describe*, much less act on. Over the years, I've learned from many projects and experiences—both successes and failures—as well as from phenomenal colleagues and peers in other companies. The job makes sense now, but I wish I'd known then what I know now.

If you've taken the staff engineer path, or are considering it, welcome! This book is for you. If you work with a staff engineer, or manage one, and want to know more about this emerging role, there'll be a lot here for you too. In the next

3 This is changing. Will Larson, LeadDev, and others have been doing phenomenal work in paving the road. I'll link to resources throughout this book.

4 I reserve the right to change my mind later on.

nine chapters I'm going to share what I've learned about how to be a great staff engineer. I'll warn you right now that I'm not going to be prescriptive about every topic or answer every question: a great deal of the ambiguity is inherent to the role, and the answer is very often "it depends on the context." But I'll show you how to navigate that ambiguity, understand what's important, and stay aligned with the other leaders you work with.

I'll unpack the staff engineer role by looking at what I think of as its three pillars: *big-picture thinking, execution* of projects, and *leveling up* the engineers you work with.

Big-picture thinking

Big-picture thinking means being able to step back and take a broader view. It means seeing beyond the immediate details and understanding the context that you're working in. It also means thinking beyond the current time, whether that means initiating yearlong projects, building software that will be easy to decommission, or predicting what your company will need in three years.[5]

Execution

At the staff level, the projects you take on will become messier and more ambiguous. They'll involve more people and need more political capital, influence, or culture change to succeed.

Leveling up

Every increase in seniority comes with more responsibility for raising the standards and skills of the engineers within your orbit, whether that's your local team, colleagues in your organization, or engineers across your whole company or industry. This responsibility will include intentional influence through teaching and mentoring, as well as the accidental influence that comes from being a role model.

We can think of these three pillars as supporting your impact like in Figure P-2.

5 Throughout this book I'm going to use the term "company" when talking about employers, but of course you could be at a nonprofit, government agency, academic institution, or other type of organization. Swap in whatever makes sense for you.

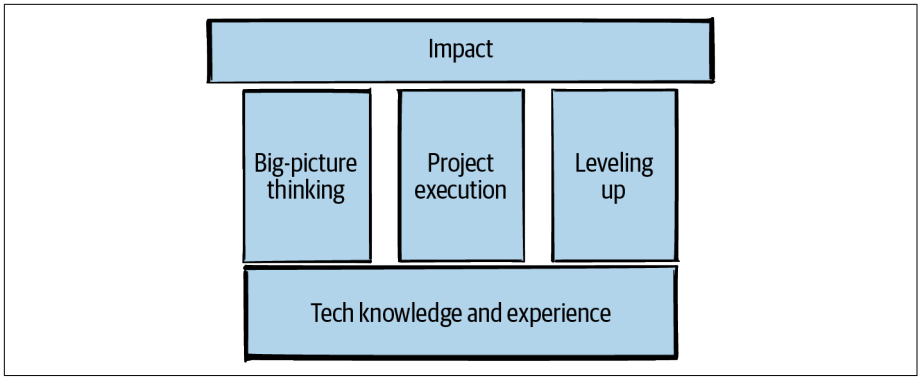

Figure P-2. Three pillars of staff engineer roles.

You'll notice that these pillars sit on a solid foundation of technical knowledge and experience. This foundation is critical. Your big-picture perspective includes understanding what's possible and having good judgment. When executing on projects, your solutions will need to actually solve the problems they set out to solve. When acting as a role model, your review comments should make code and designs better, and your opinions need to be well thought out—you need to be right! Technical skills are the foundation of every staff engineer role, and you'll keep exercising them.

But technical knowledge is not enough. Success and growth at this level means doing more than you can do with technical skills alone. To become adept at big-picture thinking, execute on bigger projects, and level up everyone around you, you're going to need "humaning" skills, like:

- Communication and leadership
- Navigating complexity
- Putting your work in perspective
- Mentorship, sponsorship, and delegation
- Framing a problem so that other people care about it
- Acting like a leader whether you feel like one or not[6]

6 And a lot more. Check out Camille Fournier's article "An Incomplete List of Skills Senior Engineers Need, Beyond Coding".

Think of these skills like the flying buttresses you see on gothic cathedrals (as in Figure P-3): they don't replace the walls—or your technical judgment—but they allow the architect to build taller, grander, more awe-inspiring buildings.

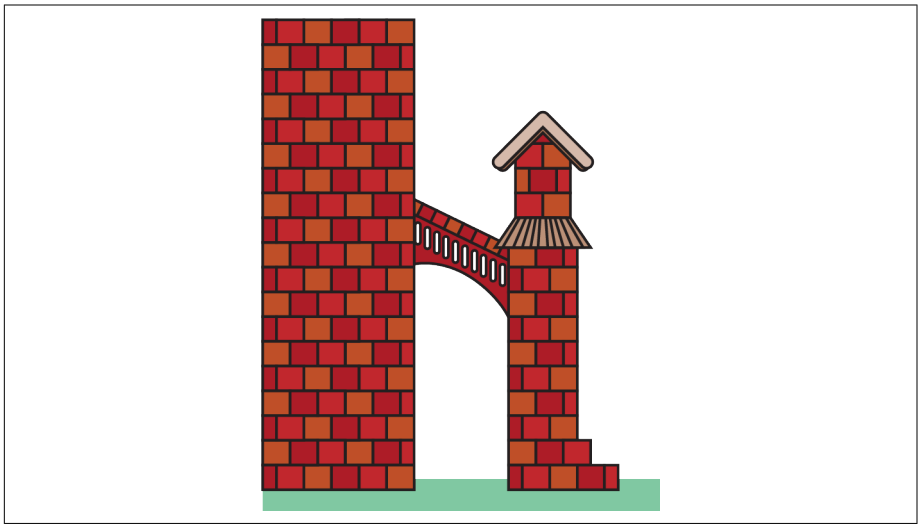

Figure P-3. Leadership skills are like the flying buttresses that let us keep massive buildings stable.

Each of the three pillars has a set of required skills, and your aptitude for each of them will vary. Some of us may be in our element when leading and finishing big projects, but find it intimidating to choose between two strategic directions. Others may have strong instincts for understanding where the company and industry are going, but lose control of the room quickly when managing an incident. Still others may boost the skills of everyone they work with, but struggle to build consensus around a technical decision. The good news is that all of these skills are learnable, and you can become adept at all three pillars.

This book is divided into three parts.

Part I: The Big Picture

In Part I, we'll look at how to take a broad, strategic view when thinking about your work. Chapter 1 will begin by asking big questions about your role. What's expected of you? What are staff engineers *for*? In Chapter 2, we'll zoom out further and get some perspective. We'll look at your work in context, navigate your organization, and uncover what your goals are. Finally, in Chapter 3, we'll look at adding to the big picture by creating a technical vision or strategy.

Part II: Execution

Part II gets tactical and moves on to the practicalities of leading projects and solving problems. In Chapter 4, we'll look at choosing what to work on: I'll share techniques for how to decide what to spend time on, how to manage your energy, and how to "spend" your credibility and social capital in a way that doesn't diminish it. In Chapter 5 I'll discuss how to lead projects that stretch across teams and organizations: setting them up for success, making the right decisions, and keeping information flowing. Chapter 6 will look at navigating the obstacles you'll meet along the way, celebrating a project that finishes successfully, and retrospecting (but still celebrating!) if it's canceled and cleanly shut down.

Part III: Leveling Up

Part III is about leveling up your organization. Chapter 7 will look at raising everyone's game by modeling what a great engineer acts like, how to learn out loud, and how to build a psychologically safe culture. We'll look at how to be the "adult in the room" during an incident or a technical disagreement. Chapter 8 is about more intentional forms of raising your colleagues' skills, like teaching and coaching, design review, code review, and making cultural change. Finally, Chapter 9 will explore how to level up *yourself*: how to keep growing and how to think about your career. Where do you go after your current role? I'll discuss some options.

One warning before we go further: this is a book about staying on the technical *track*. It is not a technical *book*. As I've said, you need a solid technical foundation to become a staff engineer. This book won't help you get that. Technical skills are domain-specific, and if you're here, I'm assuming that you already have —or are setting out to learn—whatever specialized skills you need in order to be one of the most senior engineers in your domain. Whether "technical" for you means coding, architecture, UX design, data modeling, production operations, vulnerability analysis, or anything else, almost every domain has a plethora of books, websites, and courses that will support you.

If you're someone who thinks that technical skills are the only ones that matter, you're unlikely to find what you're looking for in here. But, ironically, you might also be the person who'll get the most from this book. No matter how deep or arcane your technical knowledge, you'll find that work gets less annoying when you can persuade other people to adopt your ideas, level up the engineers around you, and breeze through the organizational gridlock that slows everyone

down. Those skills aren't easy to learn, but I promise they're all learnable, and I'll do my best in this book to show the way.

Do you want to be a staff engineer? It's fine not to aspire to more senior engineering roles. It's also fine to move to the manager track (or go back and forth!), or to stay at the senior level, doing work you enjoy. But if you like the idea of helping achieve your organization's goals and continuing to build technical muscle while making the engineers around you better at their craft, then read on.

O'Reilly Online Learning

For more than 40 years, *O'Reilly Media* has provided technology and business training, knowledge, and insight to help companies succeed.

Our unique network of experts and innovators share their knowledge and expertise through books, articles, and our online learning platform. O'Reilly's online learning platform gives you on-demand access to live training courses, in-depth learning paths, interactive coding environments, and a vast collection of text and video from O'Reilly and 200+ other publishers. For more information, visit *https://oreilly.com*.

How to Contact Us

Please address comments and questions concerning this book to the publisher:

O'Reilly Media, Inc.

1005 Gravenstein Highway North

Sebastopol, CA 95472

800-998-9938 (in the United States or Canada)

707-829-0515 (international or local)

707-829-0104 (fax)

We have a web page for this book, where we list errata, examples, and any additional information. You can access this page at *https://oreil.ly/staff-eng-path*.

Email *bookquestions@oreilly.com* to comment or ask technical questions about this book.

For news and information about our books and courses, visit *https://oreilly.com*.

Find us on LinkedIn: *https://linkedin.com/company/oreilly-media*.

Follow us on Twitter: *https://twitter.com/oreillymedia*.
Watch us on YouTube: *https://youtube.com/oreillymedia*.

Acknowledgments

Thank you to the many, many people who helped make this book a reality.

Thank you to Sarah Grey, best of all possible development editors, and all of the other phenomenal folks at O'Reilly, including acquisitions editor, Melissa Duffield; production editor, Liz Faerm; copy editor, Josh Olejarz; Susan Thompson, who created the incredible cover; and illustrator, Kate Dullea, who translated my pencil scrawls into gorgeous art. This was my first time writing a book, and you all took the terror out of it.

My thanks to Will Larson for his encouragement and support, and for helping the staff engineering community find each other for the first time. Also thanks to Lara Hogan for enthusiasm and introductions when I turned up in her DMs all "but could *I* write a book??" Thank you both for showing what sponsorship looks like.

I was lucky enough to have two of the wisest and most insightful engineers I know along on this journey. Cian Synnott and Katrina Sostek, this book is infinitely better for your review and feedback over the last year. In particular, I am indebted for your thoughtful suggestions around the parts that didn't work. Constructive criticism is always harder, and I'm grateful for your time and energy.

Many people generously shared their time to discuss ideas, offer feedback, or teach me something. I want to particularly thank Franklin Angulo, Jackie Benowitz, Kristina Bennett, Silvia Botros, Mohit Cheppudira, John Colton, Trish Craine, Juniper Cross, Stepan Davidovic, Tiarnán de Burca, Ross Donaldson, Tess Donnelly, Tom Drapeau, Dale Embry, Liz Fong-Jones, Camille Fournier, Stacey Gammon, Carla Geisser, Polina Giralt, Tali Gutman, Liz Hetherston, Mojtaba Hosseini, Cate Huston, Jody Knower, Robert Konigsberg, Randal Koutnik, Lerh Low, Kevin Lynch, Jennifer Mace, Glen Mailer, Keavy McMinn, Daniel Micol, Zach Millman, Sarah Milstein, Isaac Perez Moncho, Dan Na, Katrina Owen, Eva Parish, Yvette Pasqua, Steve Primerano, Sean Rees, John Reese, Max Schubert, Christina Schulman, Patrick Shields, Joan Smith, Beata Strack, Carl Sutherland, Katie Sylor-Miller, Izar Tarandach, Fabianna Tassini, Elizabeth Votaw, Amanda Walker, and Sarah Wells. Also, thanks to the many (so many!) other people I spoke with via DMs, email, hallway conversation, or in spirited Slack threads. You made this book better, and I appreciate you.

Thank you to the people who drink afternoon tea: you demonstrate the power of community every day. Thanks to everyone on the #staff-principal-engineering channel on Rands Leadership Slack for being relentlessly supportive and for sharing your experiences with humility. Huge appreciation to my colleagues at Squarespace and to the Google SRE diaspora. I've learned a ton from you all. And I want to thank Ruth Yarnit, Rob Smith, Mariana Valette, and the whole Lead Dev crew for the incredible technical leadership content they've shared with the world. Thanks for what you do.

Thank you to the Hillfolks, including that very good dog. I'm lucky and privileged to have you as friends. Thanks for letting me write in your caravan (and quarantine there when I had COVID!) I look forward to decades more of friendship and watching your baby oak trees grow.

Finally, to my whole family—my parents, Danny and Kathleen, and the whole extended clan—thank you for being patient over the last year while I fell off the planet.

And of course, Joel and Ms 9! I look forward to seeing you on Saturdays again. To Joel (who came up with the idea that humaning skills are "flying buttresses"), thank you for great conversations about engineering organizations and making good software. And thank you for all the sandwiches. And to Ms 9 (who was Ms 6 when I wrote the first draft of this book!): thank you for your excellent ideas, drawings, and hugs. I appreciate you lummoxes.

PART | I

The Big Picture

What Would You Say You Do Here?

The idea of a staff engineer track, or "technical track", is new to a lot of companies. Organizations differ on what attributes they expect of their most senior engineers and what kind of work those engineers should do. Although most agree that, as Silvia Botros has written, the top of the technical track is not just "more-senior seniors," we don't have a shared understanding of what it *is*. So we'll start this chapter by getting existential: why would an organization *want* very senior engineers to stick around? Then, armed with that understanding, we'll unpack the role: its technical requirements, its leadership requirements, and what it means to work autonomously.

Staff engineering roles come in a lot of shapes. There are many valid ways to do the job. But some shapes will be a better fit for some situations, and not all organizations will need all kinds of staff engineers. So I'll talk about how to characterize and describe a staff engineering role: its scope, depth, reporting structure, primary focus, and other attributes. You can use these descriptions to be precise about how you want to work, what kind of role you're looking to grow into, or who you need to hire. Finally, since different companies have different ideas of what a staff engineer should do, we'll work on aligning your understanding with that of other key people in your organization.

Let's start with what this job even is.

What Even Is a Staff Engineer?

If the only career path was to become a manager (like in the company depicted on the left in Figure 1-1), many engineers would be faced with a stark and difficult choice: stay in an engineering role and keep growing in their craft or move to management and grow in their careers instead.

So it's good that many companies now offer a "technical" or "individual contributor" track, allowing career progression in parallel to manager roles. The ladder on the right in Figure 1-1 shows an example.

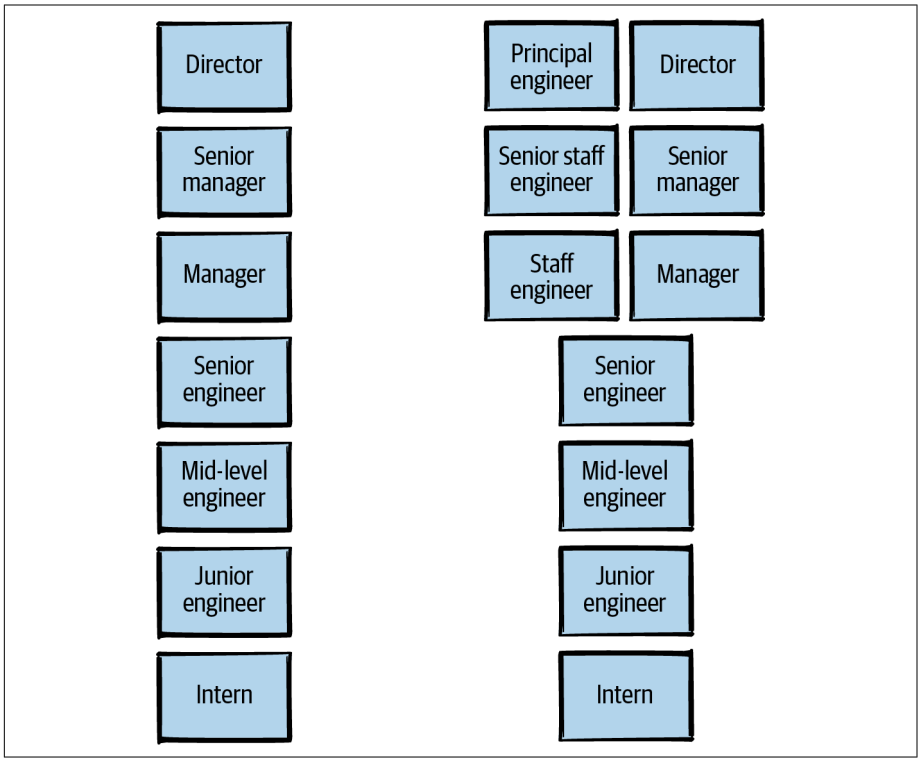

Figure 1-1. Two example career ladders, one with multiple paths.

Job ladders vary from company to company, enough that it's given rise to a website, *levels.fyi*, that compares technical track ladders across companies.[1] The number of rungs on these ladders varies, as do the names of each rung. You may even see the same names in a different order.[2] But, very often, the word *senior* is

1 I also recommend *progression.fyi*, which has an extensive collection of ladders published by various tech companies.

2 One company I heard about used the levels "senior," "staff," and "principal," in that order of seniority, but got acquired by another company that used "senior," "principal," and "staff." Chaos. The acquiring company changed all "staff" to "principal" and all "principal" to "staff," and no one was happy. Both staffs and principals saw the change as a demotion. Titles matter!

used. Marco Rogers, a director of engineering who has created career ladders at two companies, has described the *senior* level as the "anchor" level for a career ladder. As Rogers says, "The levels below are for people to grow their autonomy; the levels above increase impact and responsibility."

Senior is sometimes seen as the "tenure" level: you don't need to go further.[3] But if you do, you enter the "technical leadership" levels. The first rung above senior is often called "staff engineer," and that's the name I'll use throughout this book.

In the dual-track job ladder from Figure 1-1, a senior engineer can choose to build the skills to get promoted to either a manager or a staff engineer role. Once they've been promoted, a role change from staff engineer to manager, or vice versa, would be considered a sideways move, not a further promotion. A senior staff engineer would have the same seniority as a senior manager, a principal engineer would equate to a director, and so on; those levels might continue even higher in the company's career ladders. (To represent all of the roles above senior, I'm going to use *staff+*, an expression coined by Will Larson in his book *Staff Engineer*.)

A Note About Titles

I've occasionally heard people insist that job titles and leveling shouldn't (or don't) matter. People who make this claim tend to say reasonable things about their company being an egalitarian meritocracy that is wary of the dangers of hierarchy. "We're a bottom-up culture and all ideas are treated with respect," they say, and that's an admirable goal: being early in your career should never mean your ideas are dismissed.

But titles do matter. The Medium engineering team wrote a blog post that lays out three reasons titles are necessary: "Helping people understand that they are progressing, vesting authority in those people who might not automatically receive it, and communicating an expected competency level to the outside world."

While the first reason is intrinsic and, perhaps, not a motivation for everyone, the other two describe the effect that a title has on other

---

3 I like my friend Tiarnán de Burca's definition of senior engineer: the level at which someone can stop advancing and continue their current level of productivity, capability, and output for the rest of their career and still be "regretted attrition" if they leave.

people. Whether a company claims to be flat and egalitarian or not, there will always be those who react differently to people of different levels, and most of us are at least a little status-conscious. As Dr. Kipp Krukowski, clinical professor of entrepreneurship at Colorado State University, says in his 2017 paper, "The Effects of Employee Job Titles on Respect Granted by Customers", "Job titles act as symbols and companies use them to signal qualities of their workers to individuals both inside and outside of the firm."

We make implicit judgements and assumptions about people all the time. Unless we've invested a lot of time and energy in becoming aware of our implicit biases, it's likely that these assumptions will be influenced by stereotypes. A 2015 survey, for example, found that around half of the 557 Black and Latina professional women in STEM surveyed had been mistaken for janitors or administrative staff.

When a software engineer walks into a meeting with people they don't know, similar implicit biases come into play. White and Asian male software engineers are often assumed to be more senior, more "technical," and better at coding, whether they graduated yesterday or have been doing the job for decades. Women, especially women of color, are assumed to be more junior and less qualified. They have to work harder in the meeting to be assumed competent.

As that Medium engineering article said, a job title vests authority in people who might not automatically receive it, and communicates their expected competency level. By anchoring expectations, it saves them the time and energy they would otherwise have to spend proving themselves again and again. It gives them some hours back in their week.

The title you have now also influences the job you'll have next. Like many folks in our industry, I get daily emails from recruiters on LinkedIn. *Exactly three times* in my life I've had a cold-call recruiting email that invited me to interview for a more senior job title than the one I already had. All others have suggested a role at exactly the level that I was already at, or a more junior one.

So that's *what* the job looks like on a ladder. But let's look at *why* the technical leadership levels exist. I talked in the introduction about the three pillars of the technical track: big-picture thinking, project execution, and leveling up. Why do we need *engineers* to have those skills? Why do we need staff engineers at all?

WHY DO WE NEED ENGINEERS WHO CAN SEE THE BIG PICTURE?

Any engineering organization is constantly making decisions: choosing technology, deciding what to build, investing in a system or deprecating it. Some of these decisions have clear owners and predictable consequences. Others are foundational architectural choices that will affect every other system, and no one can claim to know exactly how they'll play out.

Good decisions need *context*. Experienced engineers know that the answer to most technology choices is "it depends." Knowing the pros and cons of a particular technology isn't enough—you need to know the local details too. What are you trying to do? How much time, money, and patience do you have? What's your risk tolerance? What does the business need? That's the context of the decision.

Gathering context takes time and effort. Individual teams tend to optimize for their own interests; an engineer on a single team is likely to be laser-focused on achieving that team's goals. But often decisions that seem to belong to one team have consequences that extend far beyond that team's boundaries. The *local maximum*, the best decision for a single group, might not be anything like the best decision when you take a broader view.

Figure 1-2 shows an example where a team is choosing between two pieces of software, A and B. Both have the necessary features, but A is significantly easier to set up: it just works. B is a little more difficult: it will take a couple of sprints of wrangling to get it working, and nobody's enthusiastic about waiting that long.

From the team's point of view, A is a clear winner. Why would they choose anything else? But other teams would much prefer that they choose B. It turns out that A will make ongoing work for the legal and security teams, and its authentication needs mean the IT and platform teams will have to treat it as a special case forever. By choosing A, the local maximum, the team is unknowingly choosing a solution that's a much bigger time investment for the company overall. B is only slightly worse for the team, but much better overall. Those extra two sprints will pay for themselves within a quarter, but this fact is only obvious when the team has someone who can look through a wider lens.

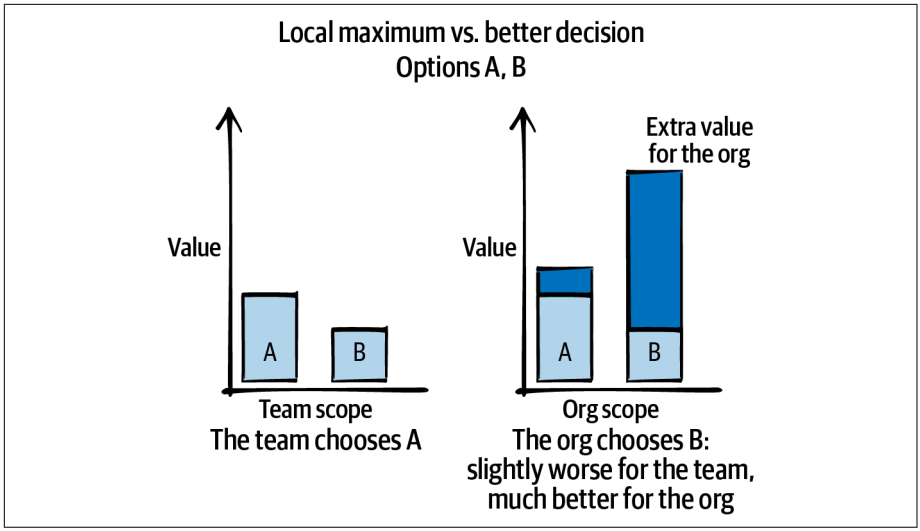

Figure 1-2. Local maximum versus better decision.

To avoid local maxima, teams need decision makers (or at least decision influencers) who can take an *outsider view*—who can consider the goals of multiple teams at once and choose a path that's best for the whole organization or the whole business. Chapter 2 will cover zooming out and looking at the bigger picture.

Just as important as seeing the big picture of the situation *now* is being able to anticipate how your decisions will play out in future. A year from now, what will you regret? In three years, what will you wish you'd started doing now? To travel in the same direction, groups need to agree on technical strategies: which technologies to invest in, which platforms to standardize on, and so on. These huge decisions can end up being subtle, and they're often controversial, so essential to making the decision is being able to share context and help others make sense of it. Chapter 3 is all about choosing a direction as a group.

So, if you want to make broad, forward-looking decisions, you need people who can see the big picture. But why can't that be managers? And why can't the chief technology officer (CTO) just know all of the "business things," translate them into technical outcomes, and pass on what matters?

On some teams, they can. For a small team, a manager can often function as the most experienced technologist, owning major decisions and technical direction. In a small company, a CTO can stay deeply involved in the gory details of every decision. These companies probably don't need staff engineers. But

management authority can overshadow technical judgment: reports may feel uncomfortable arguing with a manager's technical decisions even when there's a better solution available. And managing other humans is itself a full-time job. Someone who's investing in being a good people manager will have less time available to stay up to date with technical developments, and anyone who is managing to stay deeply "in the weeds" will be less able to meet the needs of their reports. In the short term that can be OK: some teams don't need a lot of attention to continue on a successful path. But when there's tension between the needs of the team and the needs of the technical strategy, a manager has to choose where to focus. Either the team's members or its technical direction get neglected.

That's one reason that many organizations create separate paths for technical leadership and people leadership. If you have more than a few engineers, it's inefficient—not to mention disempowering—if every decision needs to end up on the desk of the CTO or a senior manager. You get better outcomes and designs if experienced engineers have the time to go deep and build the context and the authority to set the right technical direction.

That doesn't mean engineers set technical direction alone. Managers, as the people responsible for assigning headcount to technical initiatives, need to be part of major technical decisions. I'll talk about maintaining alignment between engineers and managers later in this chapter, and again when we're talking strategy in Chapter 3.

What About Architects?

In some companies, "architect" is a rung on the technical track of the job ladder. In others, architects are abstract system designers who have their own career path, distinct from that of the engineers who implement the systems. In this book I'm going to consider software design and architecture to be part of the role of a staff+ engineer, but be aware that this is not universally true in our industry.

WHY DO WE NEED ENGINEERS WHO LEAD PROJECTS THAT CROSS MULTIPLE TEAMS?

In an ideal world, the teams in an organization should interlock like jigsaw puzzle pieces, covering all aspects of any project that's underway. In this same ideal

world, though, everyone's working on a beautiful new green-field project with no prior constraints or legacy systems to work around, and each team is wholly dedicated to that project. Team boundaries are clear and uncontentious. In fact, we're starting out with what the Thoughtworks tech consultants have dubbed an Inverse Conway Maneuver: a set of teams that correspond exactly with the components of the desired architecture. The difficult parts of this utopian project are difficult only because they involve deep, fascinating research and invention, and their owners are eager for the technical challenge and professional glory of solving them.

I want to work on that project, don't you? Unfortunately, reality is somewhat different. It's almost certain that the teams involved in any cross-team project already existed before the project was conceived and are working on other things, maybe even things that they consider more important. They'll discover unexpected dependencies midway through the project. Their team boundaries have overlaps and gaps that leak into the architecture. And the murky and difficult parts of the project are not fascinating algorithmic research problems: they involve spelunking through legacy code, negotiating with busy teams that don't want to change anything, and divining the intentions of engineers who left years ago.[4] Even understanding what needs to change can be a complex problem, and not all of the work can be known at the start. If you look closely at the design documentation, you might find that it postpones or hand-waves the key decisions that need the most alignment.

That's a more realistic project description. No matter how carefully you overlay teams onto a huge project, some responsibilities end up not being owned by anyone, and others are claimed by two teams. Information fails to flow or gets mangled in translation and causes conflict. Teams make excellent *local maximum* decisions and software projects get stuck.

One way to keep a project moving is to have someone who feels ownership for the whole thing, rather than any of its individual parts. Even before the project kicks off, that person can scope out the work and build a proposal. Once the project is underway, they're likely to be the author or coauthor of the high-level system design and a main point of contact for it. They maintain a high engineering standard, using their experience to anticipate risks and ask hard questions. They also spend time informally mentoring or coaching—or just

4 What were they *thinking*? Was this really what they intended to do? Of course, future teams will ask the same of us.

setting a good example for—the leads of individual parts of the project. When the project gets stuck, they have enough perspective to track down the causes and unblock it (more on that in Chapter 6). Outside the project, they're telling the story of what's happening and why, selling the vision to the rest of the company, and explaining what the work will make possible and how the new project affects everyone.

Why can't technical program managers (TPMs) do this consensus-building and communication? There is definitely some overlap in responsibilities. Ultimately, though, TPMs are responsible for delivery, not design, and not engineering quality. TPMs make sure the project gets *done on time*, but staff engineers make sure it's done with high engineering standards. Staff engineers are responsible for ensuring the resulting systems are robust and fit well with the technology landscape of the company. They are cautious about technical debt and wary of anything that will be a trap for future maintainers of those systems. It would be unusual for TPMs to write technical designs or set project standards for testing or code review, and no one expects them to do a deep dive through the guts of a legacy system to make a call on which teams will need to integrate with it. When a staff engineer and TPM work well together on a big project, they can be a dream team.

WHY DO WE NEED ENGINEERS WHO ARE A GOOD INFLUENCE?

Software matters. The software systems we build can affect people's well-being and income: Wikipedia's list of software bugs makes for good, if sobering, reading. We've learned from plane crashes, ambulance system failures, and malfunctioning medical equipment that software bugs and outages can kill people, and it would be naive to assume there won't be more and bigger software-related tragedies coming in our future.[5] We need to take software seriously.

Even when the stakes are lower, we're still making software for a reason. With a few R&D-ish exceptions, engineering organizations usually don't exist just for the sake of building more technology. They're setting out to solve an actual business problem or to create something that people will want to use. And

5 Hillel Wayne's essay "We Are Not Special" points out that a lot of engineering solutions that used to involve carefully tuning physical equipment are now done with a "software kludge" instead. I'm genuinely always surprised we've had so few major fatal accidents from software so far. I wouldn't like to depend on us staying lucky.

they'd like to achieve that with some acceptable level of quality, an efficient use of resources, and a minimum of chaos.

Of course, quality, efficiency, and order are far from guaranteed, particularly when there are deadlines involved. When doing it "right" means going slower, teams that are eager to ship may skip testing, cut corners, or rubber-stamp code reviews. And creating good software isn't easy or intuitive. Teams need senior people who have honed their skills, who have seen what succeeds and what fails, and who will take responsibility for creating software that works.

We learn from every project, but each of us has only a finite number of experiences to reflect on. That means that we need to learn from *each other's* mistakes and successes, too. Less experienced team members might never have seen good software being made, or might see producing code as the only important skill in software engineering. More seasoned engineers can have huge impact by conducting code and design reviews, providing architectural best practices, and creating the kinds of tooling that make everyone faster and safer.

Staff engineers are role models. Managers may be responsible for setting culture on their teams, enforcing good behavior, and ensuring standards are met. But engineering norms are set by the behavior of the most respected engineers on the project. No matter what the standards say, if the most senior engineers don't write tests, you'll never convince everyone else to do it. These norms go beyond technical influence: they're cultural, too. When senior people vocally celebrate other people's work, treat each other with respect, and ask clarifying questions, it's easier for everyone else to do that too. When early-career engineers respect someone as the kind of engineer they want to "grow up" to be, that's a powerful motivator to act like they do. (Chapter 7 will explore leveling up your organization by being a role model.)

Maybe now you're convinced that engineers should do this big-picture, big-project, good-influence stuff, but here's the problem: they can't do it on top of the coding workload of a senior engineer. Any hour you're writing strategy, reviewing project designs, or setting standards, you're not coding, architecting new systems, or doing a lot of the work a software engineer might be evaluated on. If a company's most senior engineers just write code all day, the codebase will see the benefit of their skills, but the company will miss out on the things that only they can do. This kind of technical leadership needs to be part of the job description of the person doing it. It isn't a distraction from the job: it *is* the job.

Enough Philosophy. What's My Job?

The details of a staff engineering role will vary. However, there are some attributes of the job that I think are fairly consistent. I'll lay them out here, and the rest of the book will take them as axiomatic.

YOU'RE NOT A MANAGER, BUT YOU ARE A LEADER

First things first: staff engineering is a *leadership* role. A staff engineer often has the same seniority as a line manager. A principal engineer often has the seniority of a director. As a staff+ engineer, you're the counterpart of a manager at the same level, and you're expected to be as much "the grown-up in the room" as they are. You may even find that you're more senior and more experienced than some of the managers in your organization. Whenever there's a feeling of "someone should do something here," there's a reasonable chance that the someone is you.

Do you *have* to be a leader? Midlevel engineers sometimes ask me if they *really* need to get good at "that squishy human stuff" to go further. Aren't technical skills enough? If you're the sort of person who got into software engineering because you wanted to do technical work and don't love talking to other humans, it can feel unfair that your vocation runs into this wall. But if you want to keep growing, being deep in the technology can only take you so far. Accomplishing larger things means working with larger groups of people—and that needs a wider set of skills.

As your compensation increases and your time becomes more and more expensive, the work you do is expected to be more valuable and have a greater impact. Your technical judgment will need to include the reality of the business and whether any given project is worth doing at all. As you increase in seniority, you'll take on bigger projects, projects that can't succeed without collaboration, communication, and alignment; your brilliant solutions are just going to cause you frustration if you can't convince the other people on the team that yours is the right path to take. And whether you want to or not, you'll be a role model: other engineers will look to those with the big job titles to understand how to behave. So, no: you can't avoid being a leader.

Staff engineers lead differently than managers, though. A staff engineer usually doesn't have direct reports. While they're involved and invested in growing the technical skills of the engineers around them, they're not responsible for managing anyone's performance or approving vacation or expenses. They can't

fire or promote—though local team managers should value their opinions about other team members' skills and output. Their impact happens in other ways.

Leadership comes in lots of forms that you might not immediately recognize as such. It can come from designing "happy path" solutions that protect other engineers from common mistakes. It can come from reviewing other engineers' code and designs in a way that improves their confidence and skills, or from highlighting that a design proposal doesn't meet a genuine business need. Teaching is a form of leadership. Quietly raising everyone's game is leadership. Setting technical direction is leadership. Finally, there's having the reputation as a stellar technologist that can inspire other people to buy into your plans just because they trust you. If that sounds like you, then guess what? You're a leader.

Yes, You Can Be an Introvert. No, You Can't Be a Jerk.

The idea of "being a leader" can be a little intimidating for many people. Don't worry: not all staff and principal engineers need to be "people people." Staff engineering has plenty of room for introverts—and even the quietest engineers can set a strong technical direction through their judgment and good influence. You don't have to love being around people to be a good leader. You do have to be a role model, though, and you have to treat people well.

Many of us even have stories of "that one engineer" who got shuffled into a corner because they were too difficult for anybody to deal with. The tech culture of the 1980s and 1990s, exemplified by discussions on Usenet and the like, reveled in the popular image of the difficult, unpleasant software engineer, whose colleagues not only tolerated their behavior but made weird technical decisions just to avoid dealing with them. Today, however, an engineer like this is a liability. No matter what their output is, it's hard to imagine how anyone could be worth the reduced output and growth of other engineers and the projects that fail when that engineer won't collaborate across teams. Choosing these people as role models can mess up whole organizations.