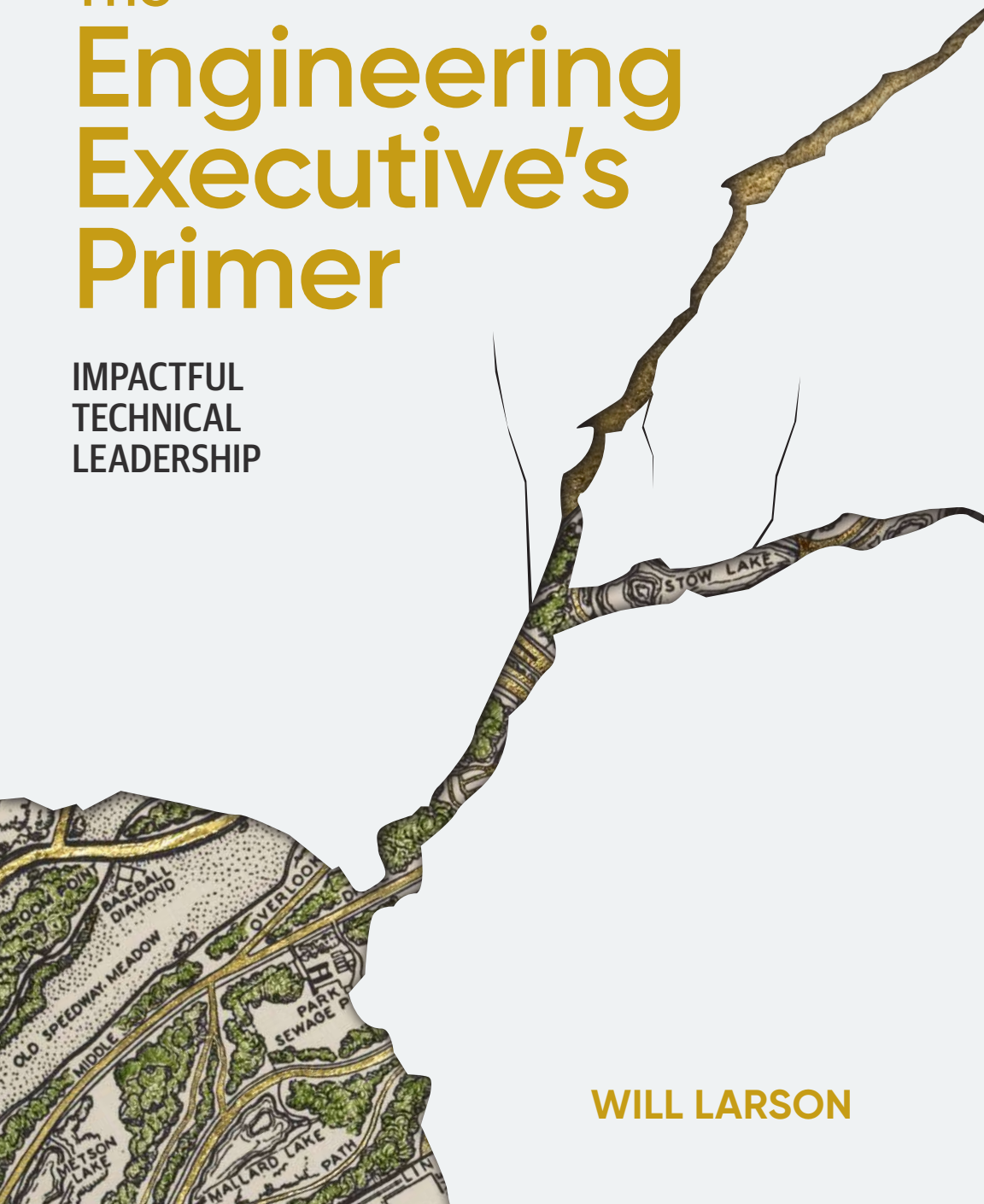
O'REILLY®

The Engineering Executive's Primer

IMPACTFUL TECHNICAL LEADERSHIP



WILL LARSON

O'REILLY®

"Will Larson's *Engineering Executive's Primer* is a must-have for senior engineering leaders looking for approachable and practical advice they can instantly deploy."

—Michael Lopp, senior engineering leader and author of *randsinrepose.com*

The Engineering Executive's Primer

IMPACTFUL TECHNICAL LEADERSHIP

As an engineering manager, you almost always have someone in your company to turn to for advice: a peer on another team, your manager, or even the head of engineering. But who do you turn to if you're the head of engineering? Engineering executives have a challenging learning curve, and many folks excitedly start their first executive role only to leave frustrated within the first 18 months.

In this book, author Will Larson shows you ways to obtain your first executive job and quickly ramp up to meet the challenges you may not have encountered in nonexecutive roles: measuring engineering for both engineers and the CEO, company-scoped headcount planning, communicating successfully across a growing organization, and figuring out what people actually mean when they keep asking for a "technology strategy."

This book explains how to:

- **Get an engineering executive job, negotiate the contract, and onboard at your new company**
- **Run an engineering planning process and communicate effectively with the organization**
- **Direct the core meetings necessary to operate an effective engineering organization**
- **Hire, onboard, and run performance management**
- **Manage yourself and remain effective through many challenges**
- **Leave the job when the time is right**

Will Larson has served as CTO at both Carta and Calm and as a software engineering leader at Stripe, Uber, and Digg. He's the author of *An Elegant Puzzle* and *Staff Engineer*. He's also a prolific writer on his blog, *Irrational Exuberance*. Before moving to San Francisco, Will grew up in North Carolina and studied computer science at Centre College in Kentucky.

EXECUTIVE LEADERSHIP

US $39.99 CAN $49.99
ISBN: 978-1-098-14948-2

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

Praise for *The Engineering Executive's Primer*

In this practical and accessible guide, Will Larson unpacks the engineering executive role with his usual clarity and insight. I learned a ton from this book—I recommend it for all engineering leaders, and for anyone who has wondered about the constraints and tradeoffs that go into executive-level decisions.

—*Tanya Reilly, senior principal engineer and author of* The Staff Engineer's Path

Will does an exceptional job of capturing the need to work on ourselves—and how to do it—while we work on people, products and strategy.

—*Julia Grace, engineering leader*

Will Larson's *Engineering Executive's Primer* is a must-have for senior engineering leaders looking for approachable and practical advice they can instantly deploy.

—*Michael Lopp, senior engineering leader and author of* randsinrepose.com

The Engineering Executive's Primer

Impactful Technical Leadership

Will Larson

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

The Engineering Executive's Primer
by Will Larson

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: David Michelson
Development Editor: Virginia Wilson
Production Editors: Jonathon Owen and Kristen Brown
Copyeditor: Nicole Taché
Proofreader: Piper Editorial Consulting, LLC
Indexer: Sue Klefstad
Interior Designer: Monica Kamsvaag
Cover Designer: Susan Thompson
Illustrator: Kate Dullea

February 2024: First Edition

Revision History for the First Edition

2024-02-06: First Release

See *http://oreilly.com/catalog/errata.csp?isbn=9781098149482* for release details.

978-1-098-14948-2

[LSI]

Contents

Preface

This is the book that I wish I'd read before starting my first Engineering executive role, and that I would have reread before starting my second executive role—to reflect on how my beliefs had evolved after doing this expansive, complex work. I hope reading this book serves you well and, more importantly, that it helps you form your own opinions rather than convinces you to take on mine.

My favorite chapter in this book is the third, which discusses creating an Engineering strategy. I also wrote a chapter about Engineering strategy in my last book, *Staff Engineer*. It's interesting how different those two chapters are, despite the fact that I wrote both of them, and only three years apart. At first, I wanted to believe this difference reflected some kind of deep insight I'd acquired between writing the two books, but the real difference is more fundamental: being an Engineering executive is a meaningfully different job, and it's forced me to adopt a different perspective than my previous roles as an engineer and engineering manager.

As an Engineering executive, you will deal with many familiar problems but will have new tools to solve them. For example, the hardest part of developing an Engineering strategy in my previous roles was usually building consensus around the solution. As an executive, the hardest part is building *conviction that your strategy is right for your company*. There are also new problems that you've probably not spent time with before, if you're new to the executive role. Everyone deals with a planning process, but only executives have to debate the algorithms to attribute platform costs across various business lines.

This book surveys the new challenges, and new tools for old problems, that you'll encounter as an Engineering executive. There are no universal answers to the most interesting questions, but you'll come away from this book with an understanding of the problems and at least one recommendation for how to approach each one.

What This Book is Not

If you're looking for details on how to run one engineering team, this won't be the most useful book for you. This book does not explore practices for running your weekly team meetings, conducting one-on-ones, or giving feedback effectively. Instead, it focuses on how multiple teams work together effectively across a company's Engineering function. For those omitted topics, I heartily recommend Camille Fournier's *The Manager's Path* (O'Reilly) and my own *An Elegant Puzzle* (Stripe Press).

Likewise, this book is focused on the whole Engineering function, which is the intersection of technology-focused and people-focused leadership. There is no meaningful way to talk about leading an Engineering function that doesn't engage with both those leadership aspects. If you're looking for a book more focused on technology-focused leadership, consider picking up Tanya Reilly's *The Staff Engineer's Path* (O'Reilly) or my own *Staff Engineer*.

Finally, this book won't be helpful if you're looking for advice on how to build a specific piece of technology. There are a thousand effective ways to build any given product, and this book won't suggest any of them. Instead, it will discuss the value of standardizing, or not standardizing, your company's approach to building and maintaining a large portfolio of products and systems. There are simply too many books out there about building technology to recommend any given one, so I'll leave you to decide what might work better for that focus.

Navigating This Book

This book is designed to be used in two different ways. If you're a new Engineering executive, or starting a new role, then you should get the most out of reading this book front to back. That will give you a broad perspective on the topics that will come up while operating in an Engineering executive role.

The second way to use this book is to come to it when you're dealing with a particular challenge, jump to the relevant section, give it a read, and put it down until you run into your next challenge. Many topics within the book are connected—what use is an Engineering strategy if you don't have a clear way to communicate that strategy to your team?—but they're all designed to stand on their own.

Clarifying Terms

To communicate more effectively, there are a handful of terms that I'm defining here to reduce potential confusion caused by inconsistent usage across companies and industry verticals:

Executive
: *This* is the functional leader of an area for the entire company. Many companies refer to certain titles as executives, for example Vice Presidents, despite them reporting to another member in the same function—a Vice President of Product reporting to a Chief Product Officer. Those Vice Presidents are not included in this book's definition of Executive.

Engineering executive
: This is the functional leader for Engineering, who is responsible for both technical execution and people management within Engineering. Depending on the company, its Engineering executive might be called a Chief Technology Officer, Vice President of Engineering, or Head of Engineering.

Team
: This describes those directly reporting to a manager.

Organization
: This describes the entire organizational chart, composed of multiple teams and their managers reporting to an executive. For example, all members of Engineering would be referred to as the Engineering organization.

***E**ngineering (uppercase)*
: This is shorthand for the Engineering organization. The teams that make up an Engineering organization will vary considerably by company. Some companies would include Product, Data Science, and Security in Engineering, and others would follow a much narrower definition. All those organizations qualify as Engineering by this book's definition.

***e**ngineering (lowercase)*
: This is the industry or profession.

As a final warning, these terms are clarified to facilitate communication, not because I'm a big believer in dogma. If someone tells you that your Engineering organization *must*, or *must not*, include any particular sub-team, or insists on a

particular definition of "executive," then be wary of their advice. There are no truths, only trade-offs, when it comes to defining terms.

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

800-889-8969 (in the United States or Canada)

707-827-7019 (international or local)

707-829-0104 (fax)

support@oreilly.com

https://www.oreilly.com/about/contact.html

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

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

There's a convenient myth that books are written solely by their authors. My experience is quite different. Technically, I wrote this book, but in practice it is the culmination of my career in engineering leadership, a decade of writing online, everything I've been taught through writing my two prior books, working closely with hundreds of engineering leaders, the direct input from dozens of industry leaders, the remarkable team at O'Reilly, and technical review by Julia Grace, Kevin Stewart, Tanya Reilly, Jasmine Tsai, and Uma Chingunde. Virginia Wilson deserves particular mention for exceptional editing and collaboration in creating this book. Finally, I simply could not have created this book without the support of my wife, Laurel, and our son, Emerson.

Getting the Job

At Digg, I ended up running Engineering, but I certainly wasn't hired to do so. It wasn't until a decade later, when I joined Calm, that a company deliberately hired me into my first executive role. If you start researching executive career paths, you'll find folks who nominally became Engineering executives at 21 when they founded a company, and others who were more than 30 years into their career before taking an Engineering executive role.

As these anecdotes suggest, there is no "one way" to get an Engineering executive job. However, the more stories you hear about folks assuming executive roles, the more they start to sound pretty similar. I've condensed the many stories I've heard, along with my own experiences, into a repeatable process that prospective candidates typically follow.

This chapter will cover:

- Deciding whether to pursue an executive role
- Why each executive job search is unique, and how that will shape your process
- Finding executive roles externally and internally
- Navigating the often chaotic executive interview process after you've gotten comfortable interviewing in well-designed middle management interview processes
- Negotiating an executive contract, particularly the terms that rarely come up in the non-executive contracts you may have negotiated prior
- Deciding whether to accept an executive offer once you have it

If you're kicking off the search for your first executive role, reading through this chapter will provide a clear roadmap through the process.

Why Pursue an Executive Role?

If you're spinning up your first executive role search, you should have a clear answer to a question you'll get a number of times, "Why are you looking for an executive role?" It's important to answer this for yourself, as it will be a valuable guide throughout your search. If you're not sure what the answer is, spend time thinking this through until you have a clear answer—maybe in the context of a career checkup.

There's no right answer, but here are some examples from real people:

- "I'm heavily motivated by learning. I've directly reported into an Engineering executive for my past two roles, and I'm looking to step into the role myself."
- "I've enjoyed working in a fast-growing company, but I also miss the direct ownership and pace of working at a small company. I'm excited to combine my previous startup experience with my recent experience at scale as an Engineering executive."

The rationale doesn't need to be particularly compelling, just something positive that expresses your excitement and qualifications for the role. Don't get discouraged if your statement isn't profound—there are very few profound ways to say that it's the next logical step in your career. Once you've written down your rationale, review it with a few peers or mentors who have already been in executive roles. Incorporate their feedback, and you're done. (If you don't have peers or mentors in executive roles, do some cold outreach to executives at companies where you've worked and see if they'll weigh in.)

The other side of this is that interviewers are also very curious about your reason for pursuing an executive role, but not necessarily for the reason you'd expect. Rather than looking for your unique story (although, yes, they'll certainly love a memorable, unique story), they're trying to filter out candidates with red flags: ego, jealousy, excessive status-orientation, and ambivalence.

One of One

Limited-release luxury items like fancy cars are sometimes labeled with their specific production number, along with the size of the overall run. For example, you might get the fifth car in a run of 20 cars overall. The most exclusive possible production run is "one of one." That item is truly bespoke, custom, and one of a kind.

All executive roles and processes are "one of one."

For non-executive roles, good interviewing processes are systematized, consistent, and structured. Sometimes the interview processes for executive roles are well-structured, but more often they aren't. If you approach these bespoke processes like your previous experiences interviewing, your instincts may mislead you through the process.

The most important thing to remember when searching for an executive role is that while there are guidelines, stories, and even statistics, there are no rules when it comes to finding executive jobs. In executive hiring, there is a selection bias for confidence. This is something that's relatively easy to find; many an executive will tell you with complete confidence how things work, but be a bit wary.

It's not just the hiring process that is not standardized; the Engineering executive roles themselves vary greatly as well. Sometimes they'll include managing Product Management, and sometimes they'll exclude managing some parts of Engineering. Working with technology-oriented founders you may provide more organizational support than technical guidance, whereas working in an older business may mean there are few other executives with a technology background. "One of one" means that anything is possible, in both the best and worst possible sense.

Finding Internal Executive Roles

Relatively few folks find their first executive job through an internal promotion. These are rare for a couple reasons. The first is that each company only has one Engineering executive, and that role is usually already filled. The second is that companies seeking a new Engineering executive generally need someone with a significantly different set of skills than the team they already have in place.

Even in cases where folks do take on an executive role at their current company, they often struggle to succeed. Their challenges mirror those of taking on tech lead manager roles, where they are stuck learning how to do their new job in addition to performing their previous role. They are often also dealing with other internal candidates who were previously their peers and who may feel slighted by not getting the role themselves. This makes their new job even more challenging, and can lead to departures that hollow out the organization's key leaders at a particularly critical time.

That's not to say that you should avoid or decline an internal promotion into an executive engineering role; just that you should go into it with your eyes open.

In many ways, it's harder to transition internally than externally. Because of that, even if an internal transition into an executive role goes poorly for you, don't assume that means you wouldn't do well as a newly hired executive at another company.

Finding External Executive Roles

Most executive roles are never posted on the company's jobs page. So before discussing how you should approach your executive job search, let's dig into how companies usually find candidates for their executive roles. Let's imagine that my defunct company Monocle Studios had been a wild success and we wanted to hire our first CTO.

How would we find candidates? Something along the lines of:

1. Consider any internal candidates for the role.
2. Reach out to the best folks in our existing network, seeing if any are interested in interviewing for the role.
3. Ask our internal executive recruiter to source candidates. (I'd skip this step if we didn't have any internal executive recruiters, as generally there's a different network and approach to running an executive search than a non-executive search; executive candidates also tend to ask different questions than non-executive candidates, which makes hiring them with non-executive recruiters even messier.)
4. Reach out to our existing investors for their help, relying on both their networks and their firms' recruiting teams.
5. Hire an executive recruiting firm to take over the search.

Certainly, not every company does every job search this way, but it does seem to be the consistent norm. This structure exposes why it's difficult to answer the question, "How do I find my first executive role?" The quick answer is to connect with an executive recruiter—ideally one that peers have worked with before—but this approach comes with some implications regarding the sorts of roles you'll get exposed to. Typically, these will be roles that have been challenging to fill for some reason.

It's important to note that the most desirable roles, and roles being hired by a well-networked and well-respected CEO, will never reach an executive recruiting firm. If you try to enter your search without an established network

and rely solely on executive recruiters to find roles, you are almost certain to be selecting from second-tier opportunities.

This is, by the way, absolutely not a recommendation against using executive recruiters. Executive recruiting firms can be fantastic. A good executive recruiter will coach you through the process much more diligently than the typical company or investor's in-house recruiter. I found my first executive role through an executive recruiter, as did the majority of my peers. (Note that the executive recruiters of tomorrow are your internal recruiting colleagues of today, so learning to partner effectively with Recruiting will pay dividends in both your current hiring and your long-term career options.) Similarly, it's not true that all founder-led searches are for desirable jobs—almost all executive roles start as founder-led searches before working their way through the pipeline.

Looking at the pipeline, there are many ways to increase your odds of getting executive opportunities at each step. The basics still matter: maintain an updated LinkedIn profile and respond politely to recruiters who do reach out. Both have a surprising way of creating job search serendipity, and ensuring your network is aware that you're looking. If you don't personally know many recruiters at investors or executive recruiters, your network can be particularly helpful for making those introductions.

There are also a small number of companies that do post executive roles publicly, and there's certainly no harm in looking through those as well. The one challenge is that you'll have to figure out whether it's posted publicly because the company is very principled about searching for talent outside their personal networks (often a good sign), or if the role has already passed unsuccessfully through the entire funnel described above (often not a good sign). Most companies with strong principles like to talk about them a lot, and you should be able to find public evidence to support whether their posting is coming from a principled belief. If you can't, then it's likely desperation.

Finally, if you're laying the groundwork for an executive search to take place a few years down the road, there's quite a bit you can do to prepare. You can join a large or high-growth company to expand your network (more on this in Chapter 12), work in a role where you get exposure to the company's investors, create more visibility of your work (more on this in Chapters 12 and 15) to make it more likely for founders to reach out to you, or get more relevant experience growing and operating an Engineering organization.

Interview Process

The interview process for executive roles is always a bit chaotic. The most surprising thing for most candidates is that the process often feels less focused or effective than their other recent interviews. This is because your hiring manager as a director of Engineering is usually an experienced engineering leader, but your hiring manager as an Engineering executive is usually someone with no engineering experience at all. In the first case, you're being interviewed by someone who understands your job quite well, and in the second, the interviewer usually has never worked in the role.

There are, inevitably, exceptions! Sometimes your interviewer was an Engineering executive at an earlier point in their career, but that usually isn't the case. A relatively common scenario in startups is when a technical founder interviews you for the role, potentially with them staying as the CTO and you taking on the VPE title. But, even then, it's worth noting that the title is a bit of a smokescreen and they likely have limited experience as an Engineering executive.

Consequently, Engineering executive interviews depend more heavily on perceived fit, prestige, the size of the teams you've previously managed, how personable you are, and how well you would navigate the specific, concrete concerns of would-be direct reports and peers. This makes the "little things" particularly important when it comes to executive interviews: send quick and polite follow-ups, use something like the STAR method to keep your answers concise and organized, prepare questions that show you're strengthening your mental model of how the company works, and generally show energy and excitement.

The general interview process that I've seen for executive roles is as follows:

1. A call with a recruiter to validate you meet the minimum requirements, are a decent communicator, and won't embarrass them if you talk to the CEO. Recruiters are heavily scrutinized on the quality of candidates they bring forward and will go out of their way to help you show up well. This is also a good opportunity for you to understand whether there are obvious issues that might make this a bad role for you, such as wrong job location, wrong travel expectations, and so forth.
2. A call with the CEO or another executive to assess interest in the role and very high-level potential fit for the role. You'll be evaluated primarily on your background, your preparation for the discussion, the quality of your communication, and perceived excitement for the company.

3. A series of discussions with the CEO or founder, in which you dig into the business and their priorities for the role. This will be a mix of you learning from them, them learning about you, and getting a mutual sense of whether you'll work well together. The exact structure of the conversations will vary depending on the CEO or founder, and will give you an understanding of what kind of person they are to work with.
4. One-on-one discussions with a wide smattering of peer executives and members of the team that you would manage. These vary widely across companies, and it is surprisingly common for the interviews to be poorly coordinated—for example, the same topics may come up multiple times across different interviewers. This is somewhat frustrating. Generally, it means the company is missing someone with the right position, experience, and energy to invest into designing the loop. I've had these interviews turn into general chats, programming screens, architecture interviews, and anything else you can imagine. All I can say is: roll with it to the best of your ability.
5. Presentation interview to the executive team, the directors, or a mix of both. Usually, you'll be asked to run a 60-minute presentation describing your background, a point of view on what's important for the business moving forward, your understanding of what you would focus on in the new role if hired, and your plan for your first 90 days.

Here are a few tips that I've found effective for these interviews:

- Ask an interviewer for feedback on your presentation before the session.
- Ask what other candidates have done that was particularly well received.
- Make sure to follow the prompt directly.
- Prioritize where you want to spend time in the presentation (on the highest-impact topics).
- Make sure to leave time for questions (while also having enough bonus content to fill the time if there aren't many).

If this sounds surprisingly vague and a bit random, then you've read it correctly. Gone are the days of cramming in all the right answers. Now, it's a matter of reading each individual effectively and pairing the right response to their perspective. If that feels arbitrary, keep in mind that navigating various perspectives will be a significant part of your role as an executive going forward!

Negotiating the Contract

Once a company decides to make you an offer, you enter into the negotiation phase. While the general rules of negotiation still apply—particularly, don't start negotiating until the company knows it wants to hire you—this is a moment when it's important to remember that these are one-of-one jobs. Compensation consultants and investors will have recommended pay ranges, but each company only hires one Engineering executive at a time, and every company is unique.

Fair pay will vary greatly depending on the company, the size of its business, your location, and your own background. Your best bet will be reaching out to peers in similar roles to understand their compensation. I've found folks to be surprisingly willing to share compensation details. It's also helpful to read DEF 14A filings for public companies, which explain their top executives' base, bonus, and equity compensation (for example, here is Splunk's DEF 14A from 2022).

There are a few aspects of this negotiation that are sufficiently different from earlier compensation negotiations:

Equity

Equity is issued in many formats: stock options, Restricted Stock Units, and so on. Equity is also issued with many conditions: vesting periods (often 4 years), vesting cliffs before vesting accrues (often 1 year), and the duration of the period after you depart when you're able to exercise options before they expire (often 90 days).

Most of these terms are negotiable in an executive offer, but it all comes down to the particular company you're speaking with. You may be able to negotiate away your vesting cliff, and immediately start vesting monthly rather than waiting a year; or negotiate an extended post-departure exercise window, even if that isn't an option more widely; or have the company issue you a loan to cover your exercise costs, which combined with early exercise might allow you to exercise for "free" except for the very real tax consequences.

To determine your negotiation strategy, I highly recommend consulting with a tax advisor, as the "best" option will depend on your particular circumstances.

Equity acceleration

Equity acceleration is another negotiation point around equity. This is worth calling out as it's common in executive offers and extremely uncommon in other cases. Acceleration allows you to vest equity immediately if certain conditions are met. Many consider this a standard condition for a startup contract, although there are many executives who don't have an acceleration clause.

One topic that gets perhaps undue attention is the distinction between single and double trigger acceleration. "Single trigger" acceleration has only one condition to be met (for example, your company is acquired), whereas "double trigger" acceleration will specify two conditions (for example, your company is acquired and you lose your job). My sense is that people like to talk about single and double triggers because it makes them sound knowledgeable about the topic rather than it being a particularly nuanced aspect of the discussion.

Severance packages

Severance packages can be negotiated, guaranteeing compensation after you exit the role. There is little consistency on this topic. Agreements range from executives at very small companies who have pre-negotiated a few months' salary as a severance package, to executives leaving highly compensated roles who require their new company to make them whole on the compensation they're leaving behind. There are also many executive contracts that don't pre-negotiate severance at all, leaving the negotiation until the departure (when you admittedly have limited leverage).

Bonus

Bonus size and calculation can be negotiated. On average, the bonus tends to be a larger component of roles outside of engineering, such as the role of a sales executive, but like everything, this is company- and size-specific. A CTO at a public company might have their bonus be equal in size to their salary. A CTO at a Series C company might have a 20% bonus. A CTO at a 50-person company might have no bonus at all.

In addition to the size of your bonus, you may be able to negotiate the conditions for earning it. This won't matter with companies that rely on a shared bonus goal for all executives (sometimes excluding sales), but may matter a great deal with other companies that employ bespoke, per-executive goals instead.

Parental leave

Parental leave can be negotiated. For example, some companies might only offer paid parental leave after one year of service, but you can absolutely negotiate to include that after a shorter amount of service. (It's worth noting that this is often negotiable in less senior roles, as well.)

Start date

Start date is generally quite easy to negotiate in less senior roles but can be unexpectedly messy for executive roles. It gets messy because the hiring company often has an urgent need for the role to be filled, while also wanting to see a great deal of excitement from the candidate about joining.

The quiet part is that many recruiters and companies have seen executive candidates accept but later decide not to join due to an opposing offer being sweetened. This creates a delay that is uncomfortable for the hiring company and for candidates who have been negotiating with other companies, including their current one.

Support

Support to perform your role successfully is another point that can be negotiated. The typical example of vain requests for support are guaranteed business- or first-class seats on business travel, but there are other dimensions of support that will genuinely impact your ability to perform your role. For example, negotiating for an executive assistant can free up hours every week for focus work, and negotiating a sufficient budget to staff your team can easily be the difference between a great and a terrible first year.

The negotiation phase is the right time to ask for whatever you'll need to succeed in the role. You'll never have an easier time ensuring that you and your organization can succeed.

Negotiate knowing that anything is possible but remember that you have to work with the people you're negotiating with after the negotiation ends. If you push too many times, you won't be the first candidate to have their offer pulled because the offering company has lost confidence that you really want to be there.

Deciding to Take the Job

Once you get an offer for an executive position, it can be remarkably hard to say no. The recruiters you're working with will push you to accept. The company you're speaking with will push you to accept. You'll have invested a great deal of work into the process, and that will bias you toward wanting to accept as well.

It's also challenging to evaluate an executive offer, because ultimately you're doing two very difficult things. First, you're trying to predict the company's future trajectory, which is hard even for venture capitalists who do it a lot (and they're solving for an easier problem as they get to make many concurrent investments, and you can only have one job at a time). Second, you're trying to make a decision that balances all of your needs, which a surprising number of folks get wrong (for example, when they take prestigious or high-paying jobs that they know they're going to hate, but just can't say no to).

I can't really tell you whether to accept your offer, but there are a few steps that I would push you to take before finalizing your decision:

- Spend enough time with the CEO to be sure you'll enjoy working with them, and that you'll trust them to lead the company. While things change a bit as companies scale, and particularly as they go public, the CEO is the person who will be deciding a company's direction, determining the members of the executive team, and taking responsibility for resolving the trickiest decisions.
- Speak to at least one member of the board. Admittedly, board members won't directly tell you anything too spicy, but their willingness to meet with you is an important signal, and it's the best opportunity to start building your relationship with the board.
- Make sure you've spoken with every member of the executive team that you'd work with regularly. Sometimes you'll miss someone in your interview process due to scheduling issues, and it's important to chat with everyone and make sure they're folks you can build an effective working relationship with.
- Have someone on the finance team walk you through the company's most recent profit and loss statement. You will need to sign a nondisclosure agreement, but it would be a very odd sign if a company was unwilling to share this information with you.

- Make sure they've actually answered your questions. I once interviewed to be a company's head of Engineering, and they refused to share their current valuation with me! I pushed a few times, but ultimately they told me it was unreasonable to ask, and I decided I couldn't move forward with a company that wouldn't even share their valuation with an executive candidate.

  Don't assume they'll disclose this information after you join the company if they won't tell you when trying to convince you to accept their offer. You will never have more leverage to get questions answered than during the hiring process: if it's important and they won't answer, be willing to walk away.
- If the company has recently had executives depart, see if you can find out why. You could learn this through mutual friends of the departed executive, or even by chatting with them directly. Sometimes you'll even have executives who interviewed you depart before, or shortly after, you join. You should *absolutely* reach out to them to understand the reasons for their departure.

As you work through these steps, ask yourself: are you still excited? Have you explained your thinking about the role to at least two friends (who didn't raise any concerns)? If the answer to these questions is yes, then take the job!

Not Getting the Job

You can't talk about running an Engineering executive search without talking about not getting the job. Who doesn't have a story of getting contacted by a recruiter who then ghosts them after an initial screen? A public company recently invited a friend of mine to interview in their CTO search. They got my friend very excited, and then notified them the following week that they had already tentatively filled the role. I've had first discussions with CEOs in which we both immediately knew we wouldn't be a good fit to work together. I've discussed roles where both I and the CEO wanted to move forward, but I lacked a specific skill they felt was required to succeed (for example, deep experience in machine learning).

Although rejection isn't fun, the perspective that I find helpful is: the goal of your search is not to find an executive job, but rather to find an executive job where you will thrive. It's much better to realize a job isn't the right fit for you before taking it, and each opportunity that doesn't move forward is for the best.

Summary

After reading through this chapter, you know that executive career searches require a different approach from the other searches you've conducted in your career. You know how to network with executive recruiters, with recruiters in venture capital firms, and, of course, with the colleagues you've worked with before. Most importantly, you've learned that every executive search is unique, and to avoid generalizing how executive searches do and don't work. Executive searches are true one-of-one experiences, and anything's possible.

Find links to further reading and resources on *https://lethain.com/eeprimer-refs-1*.

Your First 90 Days

The canonical book on starting a new role is Michael Watkins's *The First 90 Days*. If you haven't read it, you'll certainly get something valuable out of it. I did. However, you may find that its advice skews somewhat generic. Yes, you should learn as much as possible. Yes, you should achieve alignment and negotiate for resources. Yes, you should build a team and partner with other organizations.

Those points are important, and this chapter focuses one layer deeper: on what you should do in your first 90 days *as an Engineering executive*. In this chapter, I'll detail:

- What you should prioritize learning
- Why it's important to limit the number of early changes you make
- How to build trust and a support system
- Understanding your organization's health and processes
- Learning how work (and hiring, in particular) is executed
- Getting a grip on your technology when you have a limited amount of time to spend in your codebase

This chapter's structure will help you craft your initial 90-day plan for your new role, ensuring that you focus on what's most valuable rather than what happens to be screaming the loudest.

What to Learn First

Every role has a different set of priorities and different means for measuring them. What's most important for a CTO starting at a Series A company, a pre-IPO company, and Google are going to vary a lot. That said, your discovery process—figuring out what is important—requires some structure, and I'd offer up these priorities as a starting place:

1. How does the business work? Where does the money come from? Where does the money go? How much money is in the bank? What needs to be true in the next year for the business to make a step function increase in value? What are the knowledge gaps between folks in Engineering and folks operating the business? In some cases, you've never operated in the business' domain before (for example, having to learn how banking or real estate works), and in those cases it's important to dig in there as well.
2. What defines the culture? What are the company's true values? What are some key decisions that the company has made recently? How do decisions really get made? Who is valued and why? Whose role has grown and whose role has stagnated?
3. How can you establish healthy relationships with peers and stakeholders? What do your peers need from Engineering and how can you help them succeed? How do your stakeholders define success? How can you build a relationship before you encounter your first source of conflict?
4. Is the Engineering team executing effectively on the right work? How does an idea turn into finished work? How does work get assigned? Who steps in to handle emergencies?
5. Is technical quality high? How well are tools supporting the team's day-to-day work? What are the key technical behaviors and properties? What projects are considered impossible due to technical limitations?
6. Is it a high-morale, inclusive Engineering team? Who is successful within your team? Who isn't finding success? Why? What are the active inclusion efforts and who leads them? Is that work valued? What is energizing, or stealing energy from, the team?
7. Is the pace sustainable for the long haul? What do you need to stay engaged and energized for the long haul? What are lines you'll temporarily cross that you'll explicitly walk back after you've ramped up?

For each of those priorities, identify a couple of measurable goals that'll be useful in tracking your long-term progress. The act of identifying specific goals is part of the learning process, but you could imagine measuring team execution using both "number of experiments run" and the productivity metrics from *Accelerate*. Team health can be measured with skip-levels, team sentiment survey results such as CultureAmp, coffee discussions with cohorts within the team, and so on. Pace is tricky, but you can identify what you personally need for work to be sustainable and measure how frequently you're satisfying those conditions. One executive I've worked with described needing one 30-minute working block each day to stay engaged. Maybe for you it's more, or something different, and that's fine!

Making the Right System Changes

Equipped with priorities and goals, new executives often want to jump into making changes. This will become the right instinct soon but will lead you astray in early moments. Your goal as a senior leader is to make durable improvements toward these goals, which results not from just making changes but making the *right* changes. Further, durable improvements depend on building systems that create changes, not performing tactical actions that create the ephemeral appearance of improvement.

These system changes are slow to show their effects for good or ill, and your organization can only tolerate the overhead of adopting a small number concurrently. This is why to be a truly effective executive you must start by understanding the organizations and systems at play before you move toward changing them. If you skip understanding, you can emulate success in the short term, but you'll be back to working on the same problems a few months later, with less trust and feedback from the team.

If I had to create a short list of the top onboarding traps that new executives fall into, rushing to make changes before understanding the problem's shape is undoubtedly the first. The two other frequent mistakes are judging without context ("Ah, this technology is terrible, what sort of fools made this decision?") and reminiscing about past employers (the infuriating refrain of "At my last job, we...").

You Only Learn When You Reflect

Early in your career, the majority of problems you work on are difficult because they are new for you. It's challenging to do good work on problems you've never encountered before. However, the good news is that there are other folks on your team who have already experienced specific problems and understand their ins and outs.

Even for garden-variety challenges, it's easy to spend a disproportionate amount of energy trying to work through a problem before asking for help. Working hard is a platitudinal virtue, but the true virtue is *learning well*. The important outcome is becoming adept at the task; suffering along the way is optional.

I once interviewed at a company that gave new hires an explicit tool to navigate persevere/learn trade-offs: the "20-40 rule." Always spend at least 20 minutes trying to solve a problem before asking for help, and never spend more than 40 minutes before asking for help. I doubt these numbers are perfectly tuned for your team, but they're an effective mechanism to give yourself explicit permission to ask your teammates for help, while also setting the expectation that you'll spend time helping them.

The approach of working harder to overcome problems mostly works when you or someone else is managing the flow of your incoming work. Earlier in your career, your manager or a senior peer will help you manage the number of features or tickets you take on each sprint. As you get more senior, you'll increasingly be exposed to the unfiltered demands of "the business." In a slow-growing company, the increase is typically slow enough that working harder will keep up with additional work if relieved by occasional hiring. In faster-growing companies or teams, though, working harder quickly becomes a self-defeating strategy. Not only will you become too busy to teach your teammates, you'll become too busy to learn. This leads you into a downward spiral: you fall further behind the harder you work, eventually burning out.

When you're overwhelmed by a complex problem, the sort where no one has enough context or perspective to solve it for you, the only solution is to create spaces to slow down and think. You learn through reflection, especially when it comes to the most complicated problems and you have to fight your instinct to outwork these challenges.

So that's the key advice here: If you're in a rapidly changing, complicated situation—slow down! Stop working harder. There's no door in that direction. You'll never outwork deep problems, and that path is gilded with false progress, appearing to work but never getting you where you're trying to go. (This is similar to my argument against "follow the sun" on-call rotations.)

As a final thought for folks who are managing someone who is trying to outwork a complex problem: help them pause, regroup, create a space to slow down, think, and grow to overcome! There'll be new challenges soon, so save energy for the way back.

Tasks for Your First 90 Days

When you read the literature, the Platonic ideal of your first 90 days is time spent exclusively on learning and preparing for measured, effective execution on the 91st day. Depending on your circumstances, this might be feasible but it might be misguided.

If you start the learning process and uncover something deeply unwell, you have my permission to stop going down the list and instead spend your time getting that thing fixed. In that case, you might get around to some of these recommendations in nine months instead of three, and that's OK. Even if you focus on firefighting an emergency, try to avoid spending all your time on that fire. Executive leadership requires succeeding in both the short and long term, so don't get so distracted by the joy of fixing emergencies that you forgo the strategic tasks.

Another high-order bit when you consider your initial engagement is the size and complexity of the company you're joining. The complexity of a 20-person company shouldn't take you three months of learning, whereas the complexity of a 2,000-person company likely will. Like all rules, adapt them to your context.

LEARNING AND BUILDING TRUST

As you start pulling together your onboarding tasks, the first critical step is building trust within the company by meeting the folks you'll be working with, understanding how the business and company works, and learning as much as possible. Here are some actions you can take to develop your support network in the first 90 days:

Ask your manager, probably the CEO or CTO, to write their explicit expectations for you.

The classic question to ask is: "What will success look like for my role?" The answers may be very broad, along the lines of "go figure out how to be useful," which is totally fine, but if they do have more explicit expectations you should get them clearly articulated.

Figure out if something is really wrong and needs immediate attention.

Companies are surprisingly resilient, enduring wounds that at first seem unbearable. But the organization you're joining may have a dire problem, and in that case you will want to shift away from a generalized approach and focus on downgrading the situation from dire to distressed. (In some cases, folks are so familiar with the dire problem that they've forgotten it, which can be a jarring experience.)

Go on a listening tour.

If your organization contains fewer than 30 people, aim to meet with each individual privately over the first 90 days. As it gets larger, you'll need to have a mix of team and individual meetings to condense meeting more folks in the same time period. Get out of your organization, as well. Make time to meet those across the company that you won't work with as directly, using more informal opportunities like lunches.

Set up recurring 1:1s and skip-level meetings.

You'll need to establish your cadence for 1:1s and skip-level 1:1s early, as this is a key way to learn from the team. Decide on the amount of time per week you want to spend on skip-levels instead of a target frequency to keep from being overwhelmed as the organization grows. Some executives suggest waiting until after the first 90 days to schedule recurring meetings so that you can learn broadly before you learn deeply. It's important to start recurring 1:1s early, but there's some room here to experiment. When you're establishing these 1:1s, two important aspects to remember are meeting with your peers and stakeholders—not just your team—and getting to know each other as humans, not just as entwined functional gears.

Share what you're observing.

Part of earning trust in your new organization is to simultaneously show that you're listening and that you're not jumping to judgment. I've found weekly emails to your organization to be an effective format for this (more

to come in Chapter 11), but you could also use a monthly all-hands meeting or other venue.

Attend routine forums.
: Start attending existing forums and observe how they work. This works well for forums outside of your organization, too. Generally, the presence of a new executive "breaks" meetings that you're expected to start leading, such as your team's staff meeting. I find it's still helpful to ask folks to keep running them as they were for the first several iterations, but just be aware that your presence fundamentally changes the experience.

Shadow support tickets.
: Many companies have a distorted sense of how things are actually going, but customers are always a reality reservoir. The most scalable way to hear from your customers early on is to plug into support tickets. Shadow your customer success team, if you have one, as they review incoming tickets. If such a team does not exist, consider getting a Zendesk login and do some digging yourself.

Shadow customer meetings, partner meetings, or user testing.
: Exactly which of these you'll want to observe will depend on the kind of product and business you're running. The priority is learning how the company interfaces with the external folks who are essential to their success.

Find your business analytics and learn how to query the data.
: Find where your business analytics are stored, which is likely in a data warehouse, and learn how to run your own queries. You shouldn't run around assuming your queries are right—data is a subtle liar—but it's important to be able to pull your own data to perform initial explorations when you run into interesting problems. At larger companies, it's very likely you'll query data by chatting with a data scientist or business analyst rather than by directly writing queries.

CREATE AN EXTERNAL SUPPORT SYSTEM

The further you get into your career, the fewer folks at your company will have filled your current role and the more important it becomes to build a network of support that extends beyond your current team, peers, and managers. Many of the decisions that you'll make in these senior roles are slow to show results,

which makes taking advantage of others' experience even more important. Some tips for developing an external support system include:

Build an external support group of folks in similar roles.

In senior positions, there is no one else performing your role within your company, so you have to start looking externally for role models and mentors. Find a small leadership community to join, whether that's a private Slack team, a weekly breakfast group, or something else. Having a broad group to run your challenges by will vastly accelerate your learning.

Get an executive coach.

If you take the benefits of an extended support group and compact them into a small diamond, that's the experience of working with a great executive coach. They've seen or heard of a dozen others going through the same challenge you're experiencing, they understand your personal context, and your company pays them to give you feedback that you may not be comfortable hearing. If you don't know any coaches, the best way to find options is talking to industry peers.

(An ever so slightly contrarian take that one executive mentioned is that executive coaches are most helpful once you already have the working context, so you might want to engage with them a bit later as opposed to in the first three months.)

Create space for self-care.

Invest in yourself so that you have the energy to invest in others. Attend therapy, prioritize getting enough sleep, and schedule regular physical exercise.

It can be tempting to skip or postpone these actions but take advantage of the magical time when you're starting something new and your patterns are being reset. Reaching out with, "Hey, I'm starting a new role that you've been doing well for a long time, and I'd love to run a couple of questions by you!" works surprisingly well when combined with short, well-structured questions.

Managing Time and Energy

A few weeks ago, I bought a digital piano. One motivation for this purchase was that our son loves music. If you put him near a piano he will bang on, and on, and on. I've also been feeling a bit of a hole in my life, and somehow arrived at the dubious conclusion that the hole might be practicing-piano-sized. I played piano for four-ish years as a kid, never getting particularly good at it, but I can sight-read music well enough so it's been fun to pick it back up. There's something powerful in creating music through your own hands.

Buying a piano as an adult who hasn't played piano in 30 years is a celebration of hubris, but I've been enjoying it quite a bit. What's most surprising to me is that the piano creates time for itself. Playing it creates new energy for me. I'm early in the relearning curve, so it's easy to notice improvements in my playing. Speeding up my play, avoiding gaps in transition between measures, playing a simple piece from memory. Rather than draining from a fixed daily energy well, it fills the well up. This is a lesson I've already learned many times, but still manage to forget: when I'm tired, the cure is usually adding something joyful to my life; things rarely improve when I try to strip my life down toward its foundation.

That isn't to say that I haven't tried removing things. Removing things from my life is my default response to feeling anxious, so after these past three years, there isn't much left to remove: We don't even watch TV anymore since the kid was born. I've entirely reworked my home office, simplifying it down as much as humanly possible. I bought a paper shredder and shredded the miscellaneous documents I'd collected over the last 15 years. The front closet is spotless, organized, and arranged. I threw out my existing socks and replaced them with only three options: business sock, low-cut sock, and exercise sock. Now I can sort and fold my socks in a fraction of the time. That's really important, I mumble to myself with shaky conviction.

Even as I bought pianos and purged socks, I also reconnected with one of my least productive habits: searching for signs of relevance in my previous work. Any new tweets about my writing? How many Twitter (X) followers do I have? What is the sales rank for my books on Amazon? Have I updated my books' sales tracker with the last few quarters of sales? How many subscribers do my mailing lists have? I'm embarrassed

to admit that seeing these numbers go up makes me happy, but you can easily starve while eating your fill of this sort of quasi-accomplishment.

After years of trying to understand myself, I've come to know that what nourishes my particular soul is a tenuous balance between family, hobbies like the piano, and the sorts of accomplishment that I've usually found in my professional work and my writing. With a kiddo around, I've spent more time with my family, particularly with my wife, than ever before. These have been some of the most magical, rewarding, and difficult moments of my life. Even on days when our son has been sick and intermittently inconsolable, it's undeniably something special.

Things are still going well in my professional work, but executive leadership has far more indirect contributions to my happiness than do my direct accomplishments. This has been doubly true when I've been doing less hiring on my direct team and spending more time helping the organization navigate the ongoing crises of the pandemic and recession years. Where I used to have a solid chunk of writing time each Saturday, I've struggled to find any consistent writing blocks. For a while, I worried that I'd lost the ability to write, but if I have a week off work—and still have child care—then the words return quickly.

Nonetheless, I still find myself longing for more time and dreaming fondly of my life as it existed three or four years ago. It simply had so much space in it. Time that made it easy to fill so many different buckets in my life. Now, I'm learning to accept that each bucket I fill means another two will stay dry. And I'm working on adding as many small joys as I can think of. (More on this topic in Chapter 9.)

UNDERSTANDING ORGANIZATIONAL HEALTH AND PROCESS

Many in management roles are trained to avoid engaging with organizational processes because of the potential for undue friction that may prevent improvement and change. But it's essential that those in senior roles don't carry that mentality forward with them. You can't change organizational processes overnight—it takes many hands and great attention to detail to make effective organizational change—but it's essential that you're paying attention from the beginning and are able to identify any structural gaps that need addressing. Here are some ways you can better understand organizational health and processes in the first 90 days:

Document existing organizational processes.

Many organizational processes are undocumented outside of the minds operating them. Write down the processes that you run into, and ensure you fully understand them (by going through them with members of the team). This will also ease onboarding for future hires.

Implement at most one or two changes.

As you begin to understand the organization, you'll identify a long list of changes you think might help. Winnow that list down to one or two items, work with the existing team to align the items with their challenges, and then continue to evolve that process until it works.

Plan organizational growth for next year.

Most organizations are missing a clear document describing the current size, the target size for next year, and the series of translations that will need to occur to evolve the organization along the way. Write that plan. In particular, identify the critical roles that will need to be filled.

Set up communication pathways.

Set up a monthly Q&A so folks can ask you questions, start sending weekly 5–15s *(short status updates that take 5 minutes to read and no more than 15 minutes to write)* to make it easier for folks to track your attention and progress, and set up mailing lists. The details here will depend on how the company wants to communicate, but establish a few clear paths and communicate them out to your team.

Pay attention beyond the product engineering roles within your organization.

It's common for functions beyond software engineering to exist within an Engineering organization, such as TPM or SRE. Go spend time with them! These functions are always remarkably impactful and generally feel ignored. Invest time in listening to them.

Spot check organizational inclusion.

When you first get started, it's easy to spend most of your time on the folks getting the most support, which can give you a misleading sense of the organization's inclusion health. Keep your eyes open by doing a compensation review. Meet folks who work outside the office you're joining, particularly folks who work from home.

UNDERSTANDING HIRING

Most senior leaders identify hiring as their most important contribution. As a functional leader, hiring is a key focus for you in two different dimensions: personally sourcing and closing senior candidates, and structuring an effective overall Engineering hiring process.

To better understand hiring:

Track your funnel metrics and hiring pipeline.

Your hiring funnel metrics are the doorway to understanding your hiring process and where to invest in it. Getting to a place where you can review those metrics and your hiring pipeline is a key transition from a recruiting process that happens to a recruiting process that's actively improving. You'll want to get a recurring recruiting-oriented meeting onto the calendar as well, either weekly or bi-weekly, that pulls recruiters and hiring managers into a room to operate and improve together.

Shadow existing interviews, onboarding, and closing calls.

Watch the existing interview process and get a sense for how it works. Keep in mind that your presence in the room is going to change things, and tends to be less disruptive when done with senior candidates.

Decide whether an overhaul is necessary.

If you haven't previously had a strong recruiting or recruiting-oriented Engineering leader, then it's possible that there simply isn't an existing process to work from, in which case you'll have to lay out the pieces from scratch. Don't overfit on preserving what exists if even the existing team doesn't think it's worth salvaging.

Identify your (three or fewer) key missing roles.

You only have so much recruiting bandwidth and each role you're hiring for has a fixed overhead to recruit against. This means your overall hiring velocity is the highest when you focus the most, and that requires identifying a small number of key roles to hire against. If every hire is a priority, nothing is a priority.

Offer to close priority candidates.

Jumping onto closing calls for high-priority candidates is one of the earliest ways that a new executive can help the team. In addition, these are invaluable for building an understanding of your recruiting process and company brand.

Kick off Engineering brand efforts.

Part of effective Engineering recruiting is building the brand around your Engineering efforts that make folks excited to join. This is a slow process, which makes it particularly valuable to start early. This includes getting an Engineering blog up, getting the team speaking at conferences, establishing an active presence on Twitter (X), and so on—all with the goal of exciting folks to work with your team.

UNDERSTANDING SYSTEMS OF EXECUTION

Engineering's long-term value to a company is opening up new avenues of potential, but the short-term value is continued execution on the product and company roadmap. The biggest mistake a new leader can make is disrupting the current systems of execution before they have a working alternative. Instead, start with the approach of keeping what works, iterating on what doesn't, and measuring along the way. Here are some steps you can take:

Figure out whether what's happening now is working and scales.

Before getting into measurement, take a moment to understand how folks feel about the existing process and where it is or isn't working. There are so many different approaches to execution that folks often start re-creating what they've done previously rather than listening.

Establish internal measures of Engineering velocity.

I'd recommend starting with the measures from *Accelerate*, although it'll require significant tooling and process investment to make those apply in certain cases, particularly in mobile development.

Establish external measures of Engineering velocity.

It's important that some of your measures for velocity are framed in terms of business impact (revenue growth) or inputs to business impact (experiments conducted). Broad definitions keep you thinking broadly, considering how all the business' pieces fit together.

Consider small changes to process and controls.

While I don't recommend changing a lot, the two changes that I'd suggest considering are adding a weekly operational review meeting and moving into an organizational structure that cleans up any impediments to execution. If you do make a structural change, make sure that the new structure is durable for at least another year.

UNDERSTANDING THE TECHNOLOGY

Supporting, curating, and advocating for the company's approach to technology is an important part of the role. It is slightly more common for folks to engage in major technology changes after the first 90 days, but once again it comes back to identifying whether there are critical changes that need to be made now rather than later. Some actions to take to better understand the technology include:

Determine whether the existing technology is effective.
: Ask folks working with your company's various technology stacks how well the tools are serving their needs. Where are the rough edges that are stealing their time?

Learn how high-impact technical decisions are made.
: Some companies have technology reviews or architecture groups that make large or controversial technical decisions, but even more rely on informal mechanisms. Study how these processes work in your new organization.

Build a trivial change and deploy it.
: It's extremely valuable to build up your mental model of how development works within your company, and the best way to do this is by occasionally making a small change and shepherding it all the way out to production.

Do an on-call rotation.
: You probably don't have enough context to help in an incident, but you can add yourself on PagerDuty (or whatnot) as an additional page target and get a sense of the experience of being on call.

Attend incident reviews.
: Smaller companies may not yet have an incident response program, but if they do, these meetings and the subsequent reports are a uniquely valuable source of learning, and you should attend them.

Record the technology history.
: It's at best counterproductive to challenge folks on the technology decisions they made before you joined, but I do think it's useful to understand the history and factors that drove earlier decision making. Most bad decisions today were great decisions within a context that no longer exists: write down what those contexts were so that you and future hires can understand the evolution.

Document the existing technology strategy.

Surprisingly few companies have a written Engineering strategy (discussed more in Chapter 3), so document what you learn about the implicit Engineering strategy and share those notes back to the team—is this the Engineering strategy we want?

The biggest antipatterns to avoid when it comes to technology are having too many opinions or a lack of genuine curiosity about the rationale behind decisions. It's relatively low impact for early career folks to have too many opinions—folks take them with a grain of salt—but executives who express too many lightly-held opinions create a thrash in the organizations they work with, and some executives from an engineering background haven't trained themselves out of that habit.

Summary

In this chapter, you've learned how to onboard as a new engineering executive. Even with focus, there's simply a tremendous amount to learn. If the lists of tasks feel overwhelming, pick one or two areas and go deep there first. These are important areas to learn, not a checklist of things to claim you've investigated. Taking that structured approach will help you absorb the most valuable information over the course of your first three months. A more organic approach may feel better, but your goal isn't to feel busy; it's ramping up quickly to lead the organization you've just joined.

Find links to further reading and resources on *https://lethain.com/eeprimer-refs-2*.

Writing Your Engineering Strategy

Once you become an Engineering executive, an invisible timer starts ticking in the background. *Tick, tick, tick*. Eventually that timer will go off, at which point someone will rush up to you demanding an Engineering strategy. It won't be clear what they mean, but they will want it...really, really badly. *If only we had an Engineering strategy*, their eyes will implore you, *things would be OK*. For a long time, those imploring eyes haunted me because I simply didn't know what to give them. I didn't know what an Engineering strategy was.

From Appendix E of this book (which describes Stripe's aspirational approach to balancing technical standardization and exploration), to *Staff Engineer*'s guide to writing engineering strategy (which I rewrote four times from scratch), I've continued iterating on my definition. To put it simply, an Engineering strategy is a document that defines:

- The what and the why of Engineering's resource allocation against its priorities
- The fundamental rules that Engineering's teams must abide by
- How decisions are made within Engineering

Operating in the executive role, I've finally been able to solidify my point of view on what an Engineering strategy should accomplish, and how an Engineering executive can guide that strategy's creation. Reflecting on where I was a few years ago—not quite knowing what an Engineering strategy was—this chapter will walk through:

- Richard Rumelt's definition of strategy: diagnosis, guiding policies, and coherent actions
- An example of an Engineering strategy
- How and when to write your Engineering strategy
- Dealing with undocumented strategies in other functions
- How to diagnose your organization's needs and constraints
- Structuring guiding policies around your: resource allocation, fundamental rules, and how decision are made
- Maintaining the right altitude in your strategy by ensuring guiding principles are: applicable, enforced, and designed to create leverage
- The most common kinds of coherent actions in Engineering strategies
- Whether strategy should be executive-led

While the Engineering organization may contain many strategies, there is only one overarching Engineering strategy. This document—often an implicit document that no one's ever quite written down—is your constitutional document for running Engineering, and writing it is one of the most valuable things you'll do as an Engineering executive.

Defining Strategy

Richard Rumelt's *Good Strategy, Bad Strategy* is the most approachable book on strategy that I've read, and I've learned a great deal from it. Most importantly, it provides a concise, useful definition of what strategy is. According to Rumelt, a *strategy* is composed of three parts:

Diagnosis
: This is a theory describing the nature of the challenge. Its purpose is to identify the root cause(s) at play. For example, "high work-in-progress is preventing us from finishing any tasks, so we are increasingly behind each sprint" might be a good diagnosis.

Guiding policies
: These are the approaches you'll apply to grapple with the challenge. Guiding policies are typically going to involve implicit or explicit trade-offs. For example, a guiding policy might be "only hire for the most urgent team with the most urgent need; do not spread hires across all teams." If a

guiding policy doesn't imply a trade-off, you should be suspicious of it (for example, "working harder to get it done" isn't really a guiding policy).

Coherent actions

This is a set of specific actions directed by a guiding policy to address the challenge. This is the most important, and I think the most exciting, part because it clarifies that a strategy is only meaningful if it leads to aligned action.

Reading these definitions for the first time was an eye-opening experience because they answered two questions I'd been thinking about for a long time. First, how come when there is a written strategy, it's almost always irrelevant to the situation at hand? For example, I've never seen an Engineering strategy that deals with ownership of data quality across service boundaries, despite it being a topic of frequent disagreement. Second, how come so many people talk about needing strategy, but there's almost never anything written down?

The reason most written strategies don't apply is that they're actually visions of how things could ideally work, rather than accurate descriptions of how things work today. This means they don't help you plot a course through today's challenges to the desired outcome.

The reason they often aren't written down is that strategies are no more, or less, than how the organization tackles its problems, which executives often don't think is valuable to record. That's not to say that these undocumented strategies are consistently good strategies (they often aren't), but if you've ever been upset that your company doesn't have an Engineering strategy, I assure you that you do! Your Engineering strategy is how you approach your current challenges.

If you're wondering if it's too early to write your strategy down, I'd push you to make an effort. Parts of it may seem too obvious to document, but as new hires join your team, they'll find it valuable. Further into the future, it'll be extremely helpful for late joiners to read your earlier strategies to understand how your approach has evolved over time.

Example Strategy

With Rumelt's definition of strategy in mind, I've written an example strategy to demonstrate the concepts we'll work through in this chapter. If you'd prefer to end with this example, feel free to skip this section and return after reading the rest.

DIAGNOSIS

The major factors we want our strategy to address (identified by our diagnosis of our circumstances) are:

- We support three business lines (consumer, business-to-business, and new bets). About 80% of revenue comes from consumers, and 20% comes from b2b. New bets are pre-revenue. We expect the majority of revenue growth this year to occur in b2b; we expect sub-15% revenue growth in consumers; and we believe there is a small but real chance of outsized returns from new bets in the next 3–5 years.
- We are an Engineering organization of 350 (300 engineers, 40 managers, and 10 technical program managers), and project remaining cash-flow neutral to the extent that we maintain our current size.
- The number one concern in our most recent developer productivity survey is test flakiness. Further, our test stability dashboards show stability has decreased ~40% year over year, causing one in three builds to fail.
- Since growing to 350, we're seeing a significant spike in our culture survey that folks don't have the necessary information to do their work. As we followed up, we heard that people are feeling particularly unaware of releases and decisions being made on other teams.

GUIDING POLICIES

Our guiding policies to solve for those circumstances are:

- Maintain a 4:1 product engineer to infrastructure engineer ratio. This has been our ratio for the past several years; it's worked fairly well, so we intend to maintain it. For our next strategy refresh, we intend to explicitly define security and data engineering ratios, with the hope of becoming more deliberate in our staffing efforts there.
- Target 45% of our product engineering resourcing toward b2b, 35% of our resourcing toward consumers, and 10% resourcing toward new bets. The remaining 10% of our resourcing will be focused on a 12-month investment into developer productivity (specifically, this is budgeting 10% of time within existing teams to focus on developer productivity, not staffing new teams). Relative to last year, we're shifting about 20% more capacity toward b2b, as we see significant revenue growth happening there, and haven't seen growth rebound in consumers. We want to maintain an

ongoing investment toward new bets, but nothing has shown enough traction yet to warrant spinning out into its own business line. This shift will result in some team movement.

- Stay cash-flow neutral. We will avoid any actions that make us cash-flow negative. Particularly, we will avoid hiring that grows our current headcount.
- Use our standard technology stack and process (or escalate to tech spec review). We use our standard technology stack and processes (documented on our wiki) for all projects. Any projects for which we want to introduce a new technology or deviate from our standard processes should be reviewed through our tech spec review process. This allows us to ensure we maximize the impact of our developer productivity and security investments, facilitate easy cross-team transfers, and maintain our compliance controls. This policy's aim is to ensure we're placing deliberate new bets, and completing our inflight bets before making new ones. The aim is not to avoid all new bets; we encourage surfacing new ideas and putting them through the tech spec review process for consideration.
- If it's unclear and seems risky, or there's significant disagreement, quickly escalate technical issues to tech spec review, and other issues up the management chain. If there's no written decision in response to the issue, if the decision is risky or a trap-door decision, and if it's unclear who the owner is, then you should escalate! Escalations are often viewed as "negative" or "hostile," but we want to really push back on that framing. Escalations are a natural part of working together, and allow us to make quick and effective decisions with the necessary context. Escalations are only slow if we (in tech spec review or in the management team) are slow, and we should fix that by addressing latency rather than assuming that the process is inherently slow!
- All technical changes are announced in #tech-updates and all releases are shared in #shipped. This is an attempt to reduce surprises that result from technical decisions not being communicated widely, as well as cross-product impacting changes being shipped without visibility to other impacted teams. It's possible we'll need a heavier solution, but we want to start with something simple.

COHERENT ACTIONS

While many of these guiding policies are continued from last year, several of them do require specific one-time actions to implement:

- Implement organizational updates to shift product engineering from consumer to b2b. We're shifting about 10% of product engineering capacity toward b2b engineering, in support of b2b's accelerating revenue growth. This will result in several teams moving across organizations. There will be a small number of individuals moving as well, and generally we're facilitating mobility toward b2b. See details in the B2B Priorities update.
- Put in place a Test Stability working group, moving forward. As documented in the recent Test Stability update, we'll be devoting 10% of product engineering time this year toward improving test stability. We will also devote a significant amount of infrastructure engineering time there as well, with test stability being their third stability (behind security and overall site stability). Read the latest Test Stability update for more details.
- Improve our tech spec review (TSR) process: we are now also doing lightweight async reviews. We're asking tech spec review to take on a larger role in reviewing technical decisions, but at the same time we want to acknowledge that it's taking 1–2 weeks for tech spec review to provide actionable feedback on many questions coming their way. We're now asking that all requests to TSR start in chat, at which point TSR will handle them immediately and asynchronously if feasible, and ask that only complex topics come to a weekly review session. We also intend to expand staffing on TSR in the next four weeks; there will be an update on that shortly.

  If you have questions on the Engineering strategy, we'll be discussing it at our next Engineering Q&A on Wednesday, and you're also welcome to post questions in #eng-ask-anything at any time!

That ends the sample Engineering strategy, and now we'll move on to the process of writing the strategy for your organization.

Writing Process

In *Staff Engineer*, I argue that writing an Engineering strategy is like being a historian. Look at how things are already working, write them down, and share what you've written:

> *To write an Engineering strategy, write five design documents, and pull the similarities out. That's your Engineering strategy. To write an Engineering vision, write five engineering strategies, and forecast their implications two years into the future. That's your Engineering vision.*

This remains, in my opinion, the most effective way to write useful Engineering strategies as a Staff-plus engineer. That's because enforcing strategy is the biggest challenge for Staff-plus engineers driving strategy work, and this approach collects well-documented precedents to build consensus around. That consensus is the basis for enforcing the strategy going forward.

As an Engineering executive, you don't need to rely on consensus to drive enforcement, which means you can take a much more direct path to writing your strategy document. Rather than enforcement, your biggest risks are writing a weak diagnosis or ineffective guiding policies. Managing those risks leads to a different process.

The risk-management process I recommend for executives who are writing an Engineering strategy is:

Commit to writing this yourself!

Delegation is an extremely valuable executive skill, but the Engineering strategy will significantly shape how the Engineering organization functions. As the company's Engineering executive, you have the unique perspective to write this strategy. If you recently joined the company and are worried that you don't know enough yet to write it yourself, rest assured that writing the strategy is one of the best learning opportunities you'll get, and there are a number of safeguards in the process to catch mistakes.

Focus on writing for the Engineering team's leadership, both your senior-most Engineering managers and senior-most engineers.

These are the groups that will need to apply the strategy in the details of their work, and are best positioned to distinguish between a strategy that sounds good and a strategy that actually addresses their problems.

Identify the full set of stakeholders you want to align the strategy with.

While you'll certainly want to build buy-in with the full Engineering team, the stakeholders list here should be edited down a bit. It should likely include the executive team, the senior-most Staff-plus engineers, the senior-most Engineering managers, and potentially a few additional business and product leaders.

From within that full set of stakeholders, identify three to five who will provide early, rapid feedback.

This is your strategy working group. You'll share your roughest drafts with them, and their input will deeply shape the document. I recommend several Staff-plus engineers, a few of your direct reports, and your product counterpart. At earlier stage companies, this may well include the CEO, but remember that you're here to do something the CEO isn't able to do (whether it's due to time or capability constraints), so I'd generally steer away from including the CEO in this smaller working group.

Write your diagnosis section.

Start with the current roadmap, competitive pressures, and financial plan. If you have cultural survey or developer productivity survey data, incorporate those results. Pull in all the problems you've learned of in your 1:1s and team meetings. Remember to trust your judgment: you're leading the Engineering function for a reason. However, don't trust your judgment too much. Once you've drafted the diagnosis, workshop it with the individuals in your strategy working group. I'd generally recommend doing it 1:1 with each member, as it's easier to get direct feedback in a smaller group. Meet with one group member, listen to their feedback, then incorporate that feedback before reviewing it with the next member. Once you've incorporated all that feedback, share a final draft with the working group before moving forward.

Write your guiding policies.

This starts with the same process as writing your diagnosis, but it ends with an extra step: after incorporating working group feedback, I highly encourage you to also get private feedback from two to three external Engineering executives. Feedback from other Engineering executives, who understand your incentives and challenges, is uniquely valuable. Keep in mind that external executives won't have a lot of your company's context, something that will color their advice.

Now share the combined diagnosis and guiding policies with the full set of stakeholders.

I recommend sharing the full document with the group, then spending time with those who have the most feedback. You often won't get much feedback at this stage, which is fine. Part of the reason for sharing the plan at this stage is to start socializing it rather than simply crafting it. Be aware that once you share the document with the full set of stakeholders,

it will almost inevitably leak out to the wider company. Companies are designed to spread context, not to conceal it, and you just can't fight that tendency. Instead, make sure to edit out anything too sensitive to share widely, particularly changes that directly impact individuals or teams.

Write the coherent actions.

The actions are usually less complex and less controversial than the guiding policies themselves, although not always. Iterate on the draft actions with the working group. If the actions are controversial, you may want to review them with some members of the extended stakeholder group, but often that's not necessary.

You're almost ready to share the strategy with the full organization, but there's one step left:

Partner with your working group to identify the individuals in the wider team who are most likely to be upset or to strongly disagree with the strategy.

Then go spend time 1:1 with those people. Your goal is to make sure they feel heard, and that you understand their feedback. If the feedback is helpful, then incorporate it into the strategy, but be careful not to compromise the strategy to make it more popular.

Share the written strategy with the Engineering organization.

Schedule a meeting to share the strategy and the rationale behind it (as well as take questions) and establish a timeline for taking feedback. Try to keep this timeline short, roughly a week or so. An extended feedback timeline generates more feedback but rarely better feedback and prevents you from taking advantage of the strategy.

Finalize the strategy, send out an announcement, and commit to reviewing the strategy's impact in two months.

Your Engineering strategy is complete (for now; you'll of course be updating it on at least an annual cadence).

Although there are a number of steps here, they can be done very quickly, and the process is much faster than the Staff-plus engineer's documentation-driven approach. Not only is this task relatively quick, it's one that I'd recommend Engineering executives start on sooner than later.

WHEN TO WRITE THE STRATEGY

You can work your entire career without seeing a documented Engineering strategy. When I worked at Uber, there were many rules scattered across the organization, but there was never a documented, overarching Engineering strategy. Stripe, similarly, had no unified Engineering strategy, although there were numerous technical jurisdictions, such as detailed requirements for external APIs.

Shortly after joining Calm, I opened a document, titled it "Engineering Strategy," and then stared into the blank abyss until filing it away. A year later, I came back and documented three core, guiding principles: choose boring technology, resolve conflict with curiosity, and prefer vendors for commoditized functionality. These few, simple statements greatly eased decision making, allowing us to focus more time on improving our product and business. (If I could go back in time, I would move "resolve conflict with curiosity" into our values rather than including it in our strategy, but it's what I wrote at the time.)

In all three cases, the organizations would have benefited from having written a technology strategy sooner. The three questions to ask yourself before getting started are:

Are you confident in your diagnosis or do you trust the wider Engineering organization to inform your diagnosis?

> If you're not confident in your diagnosis and you're still not sure whose judgment to trust, then it's too early to move forward with writing a strategy.

Are you willing and able to enforce the strategy?

> If you think that engineers will successfully escalate past your strategy to the CEO, or that you'll be unwilling to enforce the strategy if teams ignore it, then it's too early to work on strategy.

Are you confident the strategy will create leverage?

> Strategies require an entire organization to change its behavior, which is quite expensive, and an ineffective strategy will quickly wear down people's trust in you. If you don't have at least a year's worth of conviction that it'll work, then it's not worth solidifying.

If your answers to those three questions are fairly positive, then I'd push to document the strategy now rather than next month. While it's reasonable to be wary of rolling out strategy too early, every bad strategy rollout I've seen has been

rooted in the executive's inability to listen to clear feedback. Waiting wouldn't have improved their strategy anyway. Even at its worst, a documented strategy facilitates conversations that lead to an improved strategy. You should push to have a strategy written within the first six months of starting a new role.

Dealing with Missing Company Strategies

Engineering strategies are rarely documented, and unfortunately this is part of a larger problem: surprisingly few companies document any of their strategies. This is a problem because it's remarkably difficult to write an effective diagnosis without understanding other company strategies that should inform Engineering's strategy, as captured in the overlapping circles in Figure 3-1.

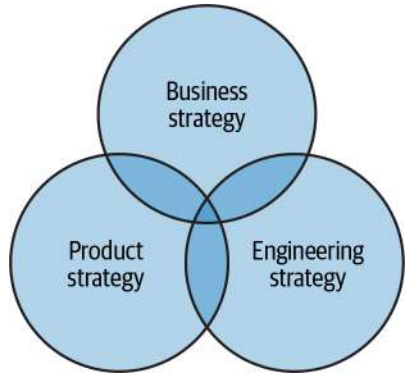


Figure 3-1. Three intersecting company strategies

The good news is that these strategies *do* exist, even if they aren't written down or are visions pretending to be strategies. The bad news is that you'll have to do some work of your own to make sure you understand these other strategies before you start writing your strategy for Engineering.

The overachieving executive's default approach here is trying to train the company on writing good strategies, and then running a company-wide strategy documentation process. That might work but going that route will slow down writing a useful Engineering strategy by months, quarters, or years. You can probably move faster by modeling a good strategy within Engineering than trying to directly change the overall approach to strategy documentation; if the executive team was aligned on doing strategy this way, it would already be done.

Instead, I recommend focusing on the handful of non-Engineering strategies that are most relevant to Engineering, and privately documenting their strategies

yourself. The Business and Product strategies are usually the most important starting places, but that will vary by the particulars of your business. For each strategy you need to document, spend time with the responsible executive to understand their perspective until you can write a short draft of their strategy. Once you have the draft, share it with that executive to make sure it's generally accurate, then keep it as a private draft to inform your understanding. It may be tempting to share this written draft more widely, but instead of unlocking progress on the Engineering strategy, you'll instead find yourself in conflict with an executive for undermining them.

Some of the questions that I've found valuable to explore in these draft strategies are:

- What are the cash-flow targets?
- What is the investment thesis across functions (e.g., sales and marketing, research and development, general and administrative)?
- What is the intended role of mergers and acquisitions?
- What is the business unit structure? How do the business units support one another? How are costs expensed across business units?
- Who are your products' users, what do they need, and how are you prioritizing in relation to those users?
- How will other functions evaluate success over the next year?
- What are your current distribution mechanisms, and how are you trying to change them?
- What are the most important competitive threats?
- What about the current strategy is not working?

Drafting these missing strategy documents is always tricky, and your goal is to pull together a reasonable sketch, not to write something perfect. If you get stuck, spend another cycle validating your diagnosis and then move forward to guiding policies. If you've missed something important, writing the policies will often reveal the gap for you.

Establishing the Diagnosis

Your diagnosis is your strategy's foundation. If you get the diagnosis wrong, your guiding policies and actions will focus on the wrong problems. Nonetheless, I find most strategy writers try to skim past the diagnosis section, which usually

leads to an underpowered strategy, even if they spend a great deal of time on subsequent sections.

A few pieces of hard-earned advice on writing an effective diagnosis:

Don't skip writing the diagnosis, even if it's very tempting to do so.

By the time you start writing a strategy, you've often settled on a handful of guiding policies that you want to adopt. Maybe you're convinced that you should cancel your in-flight migration to microservices. A word of caution: don't back into your diagnosis. If you don't start with your diagnosis, you can't evaluate whether a solution is appropriate. More importantly, it's hard to convince anyone else that your approach makes sense because they will have their own implicit diagnosis that likely won't quite be the same as yours. Conversely, when you start with writing a diagnosis, that is the easiest place to build alignment!

When possible, have two or three leaders diagnose independently.

An effective way to identify gaps in your diagnosis is to have a few leaders independently write a diagnosis section. This doesn't need to be perfect—even a rough draft will usually identify gaps among the leaders' perspectives. This is especially true when the leaders operate in different roles—for example, a Staff-plus product engineer, a Staff-plus infrastructure engineer, and a senior Engineering manager working in different business lines. Further, this approach reduces the risk that you, as a new leader, are missing a great deal of relevant context.

Diagnose with each group of stakeholder skeptics.

Validate your diagnosis across a variety of groups. Talk to engineers, Engineering managers, leaders in peer functions like Product, and the executive team. It's particularly useful to identify skeptics within those groups who are prone to disagreeing. You shouldn't accept the skeptics' concerns at face value, but you should listen to where they raise concerns. Come back to those areas to confirm whether you are convinced with your analysis, or if you should develop a firmer diagnosis.

Be wary when your diagnosis is particularly similar to that of your previous roles.

Leaders in a new role often possess an unfounded belief that challenges in their new role will largely parallel the challenges in their previous roles. Many leaders cling to their belief, even when it's clearly untrue (for example, building highly scalable infrastructure makes sense at companies with

strong product-market fit, but probably doesn't make sense for companies without a product). If your diagnosis is strikingly similar to the diagnosis you wrote for your previous role, make sure that you're anchoring on your new reality rather than memories.

There's one other understated value of writing an effective diagnosis: it makes it clear to readers that you're committed to your strategy, and acknowledging and resolving the issues at hand. Often those issues are uncomfortable to acknowledge in writing, and the bravery (and the tact) to nonetheless address them directly will build more conviction in your strategy than anything else.

Structuring Your Guiding Policies

With your diagnosis in hand, the next step is determining guiding policies. There are many, many ways to articulate your guiding policies, but I would recommend starting by answering three key questions that I believe get at the heart of effective Engineering strategy:

What is the organization's resource allocation against its priorities? (And why?)

Competition can be healthy, but competing internally on budget and headcount tends to reward empire building rather than effectiveness. It also means you'll often underinvest in critical priorities like compliance or security. Avoid this sort of internal competition by ensuring your Engineering strategy clearly articulates resourcing and priorities.

As an Engineering executive, it's particularly important to think about the priorities that no one else is asking for (especially security, reliability, compliance, and developer productivity), and to ensure your investment thesis addresses those.

Just as important is connecting the resource allocation back to your diagnosis. This grounds your allocation in the specific constraints you're solving for and makes it clear what problems any counterproposal must address.

Example: We aim to maintain a ratio of four product engineers for every one platform engineer (e.g., security, reliability, infrastructure, developer productivity). In addition to that standard ratio, this year we are running two major projects outside of that ratio, prioritizing a total of 10 engineers on security (all production access requires MFA and is connected to an uneditable audit trail), and developer productivity (progressive migration of all JavaScript codebases to TypeScript).

What are the fundamental rules that all teams must abide by? (And why do they matter?)

Many of the most impactful guiding policies are predicated on broad, consistent adoption. For example, requiring all backend projects to be implemented in Golang would greatly narrow your security, compliance, and tooling needs. Similarly, requiring all new projects to use a specific database would also narrow those needs.

These sorts of rules must be specified at the Engineering organization level because that's the only place where you can make the appropriate, organization-level trade-offs.

Folks are much more open to following rules if you explain why the rules are valuable, so I strongly recommend you explicitly explain why each rule is important. Things that are obvious to you may not be obvious to others.

Example: All development must use our standard development stack (background services use Golang, frontend services use TypeScript, storage is in a service-isolated instance of Aurora PostgreSQL) and our development lifecycle (standard code review, linting, and deployment processes documented in the Development Lifecycle wiki). Exceptions to these rules must be approved by both tech spec review and the CTO.

How are decisions made within Engineering? (And why do we work this way?)

Even the most comprehensive strategy will omit many important details, but it should explain how decisions are generally made. Think of it as a nuanced navigation of positive and negative freedoms between teams and others impacted by their decisions.

You want teams to know what they can decide themselves, what they should optimize for when making those decisions, and how to move forward with decisions that they can't make independently. You also want individuals to know why you work this way: there are many implicit trade-offs in each way of working, and these trade-offs are often invisible to folks who are frustrated with the current process.

Example: Technical decisions that deviate from the standard development stack or standard development lifecycle should be approved by tech spec review and the CTO. Changes to those two standards should similarly be approved by tech spec review and the CTO. Changes to organizational structure, hiring prioritization, and general people processes should be approved by the CTO. All other decisions should be made by the teams

and leaders closest to the decision. If anyone believes we are making a meaningfully suboptimal decision, please escalate that decision using our Escalation Process.

If you can answer those three questions clearly, you will have an uncommonly valuable Engineering strategy. At least as important, the strategy will be explicit about how it ties into the surrounding company strategies, and the degree of freedom it cedes to the teams and leadership within Engineering.

Conversely, if your diagnosis doesn't support answering these questions, then I'd push you to think more deeply about your diagnosis. It's likely accurate but missing an altitude that an executive is uniquely suited to bring.

Maintaining Your Guiding Policies' Altitude

It can be difficult to write guiding policies without unnecessarily constraining the teams within your organization. You can argue that each team's roadmap and technology choices fall within the scope of Engineering strategy. I recommend using Engineering strategy sparsely, while ensuring you take advantage of its unique benefits.

To ensure your strategy is operating at the right altitude, ask if each of your guiding policies is applicable, enforced, and creates leverage:

Applicable: It can be used to navigate complex, real scenarios, particularly when making trade-offs.

Much as applicability is essential for useful values, it applies to guiding policies as well. Guiding policies should be living, useful tools. If you can't apply them, then scrap them!

Example: We generally prioritize stability of the existing product over new product work. If stability work takes less than a week, teams should self-approve the work. If it takes longer, they should review sequencing one step up their management chain.

Example: We prefer SaaS vendors over building our own commodity solutions, but we only consider SaaS vendors with current SOC 2 Type 2 compliance. Build-versus-buy decisions should be reviewed by tech spec review. Exceptions to our SOC 2 Type 2 policy should be approved by the CTO (but won't be granted).

Enforced: Teams will be held accountable for following the guiding policy.

Guiding policies will only actually guide an organization if they're enforced. Every experienced engineer has their own stories of working somewhere with a standardized technology stack, hiring a new engineer who doesn't want to use it, and the ensuing conflict. A policy is only effective to the extent that you are willing to enforce the policy, even if the person violating it is your friend, or previously worked at a cool company.

It's hard to talk about universal examples here. Instead, this is more of a cultural question for you to ask yourself: are you willing to enforce this policy? If not, look for something else that you're willing to enforce. Often the gap between unenforceable and enforceable can be bridged by a simple nuance (such as "unless approved by the CTO").

Creates leverage: Create compounding or multiplicative impact.

Leverage is making the organization more efficient, either directly (e.g., using a data interface that abstracts data privacy issues from product engineers) or indirectly (e.g., creating a new machine learning-powered content selection tool, which means folks don't need to argue about what content is shown where).

Many forms of leverage are accessible to the teams within Engineering, and it's often not necessary to directly address those opportunities in your Engineering strategy. However, some approaches must be deployed at the Engineering strategy layer to be impactful, particularly standardization strategies that require organization-wide commitment (e.g., everyone uses TypeScript for frontend development).

Your Engineering strategy also needs to solve for scenarios in which no team is capable of prioritizing a given effort, despite the effort being very valuable, such as a compliance or privacy initiative that doesn't fall cleanly into any given team's scope but is necessary for continued business operation.

Example: Google historically constrained development to four languages: Python, C++, Go, and Java. They enforced this fairly rigorously and it created leverage in their development tooling. Each new project happening within that ecosystem increases a centralized tooling team's impact on the company.

A closely related example is Dan McKinley's "Choose Boring Technology", a strategy that advocates building leverage by constraining technology

choice, which was heavily enforced during Kellan Elliott-McCrea's era of Etsy Engineering leadership.

Example: Uber (in 2014) had an implicit technology strategy, related to its "Let builders build" value: letting teams select their own tools. This aimed to create leverage by allowing teams to select the best tool for the job at hand and was enforced through both Engineering leadership's absence of an enforced counter-policy, and the use of a permissive service tool that merrily ran any Docker container.

This approach is implicitly grounded in the theory that teams' individual gains will outweigh the inability to operate a high-leverage developer productivity team. More importantly, it highlights the value of explicit Engineering strategy, versus implicit, inconsistently applied Engineering strategy, which is often ineffective. While ineffective in Uber's case, at a consulting company that built bespoke tools for other companies, it's possible this guiding policy would be very effective.

If one of your guiding policies doesn't meet these criteria but is necessary to address your diagnosis, then I wouldn't worry about it. However, if you find that many of your guiding policies don't meet these criteria, then it's worth spending time reflecting on what's creating that gap until you're confident that they do meet these criteria, or that these criteria don't apply to your diagnosis.

Selecting Coherent Actions

The final component of your Engineering strategy is a set of coherent actions to implement your guiding policies. If you follow my recommended approach to structuring your guiding policies, then you'll find three major categories of coherent actions:

Enforcements

How will Engineering's rules be maintained? Even the most thoughtful policies won't create leverage if they aren't followed. Enforcement actions explain the process used to maintain policies, as well as a clear, ongoing process for evaluating exceptions. Although the formality here may feel awkward, it's much easier to be explicit.

Example: Tech spec review will meet weekly and review requested exceptions to our standard development stack.

Escalations

How should folks constructively argue that a guiding policy doesn't work or doesn't apply to their case? Ideally, there are just a few (or even just one) escalation processes shared across all guiding policies. Many people view "escalations" in a very negative light, but I'd push you to rethink that perspective. Whether you acknowledge them or not, escalations are going to happen implicitly, so it's better to acknowledge and structure that process.

Example: If you believe our guiding policy is meaningfully wrong, escalate up the technical or managerial reporting chain.

Transitions

How do we transition between the current state and the new state? This is particularly relevant to changes in resource allocation, which might involve people or teams changing their focus. In its simplest form, this might be a few people shifting focus for a quarter. For a larger shift, it might be an Engineering reorganization.

In cases in which your guiding policies require a transition from one technical approach to another, such as changing your primary data storage technology, then the action is conducting a migration.

Example: We are winding down our proposed migration to services, and instead recommitting to our monolithic service. Team structure won't be impacted, but priorities will be. First, individual teams will stop work on service migration. Second, our developer productivity team will make stable builds their top priority.

These actions are a bit unusual because Engineering-scope guiding policies should persist for many months or even years. Long-lasting guiding policies require fewer one-off actions, and more enduring actions to maintain the policies over time. It's natural, and even desirable, to feel a bit anxious that your Engineering strategy's actions aren't action-y enough. As long as they clearly connect back to your diagnosis, then you're in good shape.

Shouldn't Strategy Be Bottoms-Up?

Even with that efficiency argument in mind, there's a certain type of person who encounters a strategy and immediately argues that it's disempowering. The argument comes in a couple different flavors:

- Strategies are top-down, removing autonomy from teams doing the work. We value autonomy, so teams should determine their own strategies.
- Managers, including the Engineering executive, shouldn't set strategy; it should be owned by engineers rather than managers.

I think these arguments misunderstand how strategy works. Strategy can only be employed top-down, so it's not a question of top-down versus bottoms-up. Instead, it's a question of whether to have strategy at all. Outside of small, slow-growing organizations that can use social pressure to enforce strategy, only top-down leadership (or groups wielding that delegated authority) are capable of enforcing a practice, which is necessary for an effective strategy.

When I hear arguments like these, usually the concern is really about how strategy is being set (e.g., did our migration to Go ignore certain perspectives?) or with a specific strategy (e.g., is our monorepo strategy not working?). If you can find the root concern, there's almost always something interesting to learn there!

If someone is genuinely opposed to a consistent organizational approach, then they're opposed to most scalable forms of leverage. Certainly, you can successfully run a small company that way but it's an expensive way to operate anything larger, and that's a lesson for them to learn rather than your organization to suffer through. (As always, some caveats exist. If your company only does prototypes that never go to production, or if you always contract out to build for other companies, then you'd be able to offload most of the costs to them instead, but I'm skeptical if that's the right choice in most cases.)

Summary

Strategy is a deep topic, and it's easy to drown in that depth if you aren't careful. This is why so many executives never end up writing anything. They start working through the diagnosis, guiding principles, actions, application, enforcement, and creation of leverage. Then they get distracted by a new problem. Or what they've written feels a bit too obvious to bother sharing. Or they have an upcoming decision to make and want to pause releasing the strategy until they can include that decision as well. These reasons cause them to push the strategy off until tomorrow. Then next week. Then indefinitely.

If you look at the full set of ideas and practices here and think, "I'm not going to do all that," then I think that's fine. This doesn't have to be a massive, one-time creation. I firmly believe you can be a top 10% Engineering strategist by simply documenting your existing implicit strategies. Once you've

documented them, discussions will happen more frequently, which will create a persistent pull toward improvement. It's better to go slowly than to avoid starting altogether.

Find links to further reading and resources on *https://lethain.com/eeprimer-refs-3*.

How to Plan

Some years back, I interviewed a senior leader for an engineering role, and asked them a question about planning. I enjoyed their response, "Ah yes, the 'P' word, planning." That answer captured an oft heard perspective that planning is some sort of business curse word. Even when it goes well, planning is an objectively difficult task. When it goes poorly, the business loses months or years of potential progress.

James Carse's *Finite and Infinite Games* (Free Press) proposes that you can view most things in life from two different perspectives. The first is seeing life as a series of finite games, with clear rules and specific ways to win. The second is seeing life as an infinite game, with rules that are evolved over time by its players, and where the goal is continuing to play. Although planning is often viewed as a finite, rules-heavy process, I find it much more useful to approach planning as an ongoing game with dynamic rules. It was only when I was able to look past the stated rules that I was able to become an effective planner.

Guiding your company through the dynamics of planning is uniquely impactful executive work and is an area where effective executives can greatly distinguish themselves. In this chapter, we'll walk through:

- Discussing the default planning process at most companies
- Dividing planning into three discrete phases: financial plan, functional portfolio allocation, and roadmap
- Establishing a shared timeline across the three phases
- Exploring planning's frequent failure modes

By the end, you'll be prepared to operate effectively within an existing planning process, balance functional and cross-functional demands, and diagnose how your current planning process may be holding Engineering, and your company, back.

The Default Planning Process

By the time companies reach a couple hundred employees, they mostly reach the same documented planning process:

- Every year, the executive team agrees on an annual financial plan, including a headcount plan.
- Every quarter (or half), the executive team will create a planning artifact that describes the plan for the upcoming quarter—such as objectives and key results (OKRs) for each team, target business outcomes, or a list of projects to deliver.
- Teams are responsible for managing their own execution against the quarter or half plan, using the process of their choice (e.g., Scrum, Agile).
- There may be a monthly execution review—such as a business review—to track progress against company goals.

This process isn't entirely uniform across companies—details about when the process happens, what document you need to deliver to whom, or how budgeting works could vary. What *will* be particularly different across companies is how planning *actually* works. While the executive team will nominally follow the planning pattern outlined here, there is always another layer of emergent rules based on their behavior. For example, some companies have a stated financial planning process in which the executive team creates a shared annual financial plan, but in practice, individual executives privately advocate to the CEO to increase their function's headcount. If the CEO rewards those private escalations, then that becomes the *real* process and the jointly created financial plan will become a polite fiction.

Even in companies where the executive team works together in good faith, new information may emerge that invalidates the current plan:

- A country where you do business might pass an unexpected privacy regulation.
- A financial partner might go out of business.
- You might settle an accessibility lawsuit with a discrete timeline to make improvements.

From the finite game perspective, updating your plan to address these new concerns is a failure, but from the perspective of operating a well-run business, you obviously want to operate against a plan that addresses today's reality rather than last month's.

Planning's Three Discrete Phases

Without shared goals for the planning process, planning often becomes a bloated, overloaded system that poorly solves many problems. You will encounter planning processes that spend the majority of their time on whether you've completed security reviews, ensured you're staffing cross-functional projects, or verified that headcount planning is aligned with recruiting capacity. Those are all valuable, but they're solvable outside of the planning process and distract from where planning can be most valuable.

Your planning process will be effective to the extent that it can do three things well:

Set your company's resource allocation across functions, as documented in an annual financial plan.

This plan sets targets for revenue and expenses, broken down by function and business line. From this you can answer questions such as the relative investment into Research & Development (R&D) versus Sales & Marketing (S&M) or General & Administrative (G&A), as well as among the company's products or business units.

Refresh your Engineering strategy's resource allocation, with a particular focus on Engineering's functional portfolio allocation between functional priorities and business priorities.

> There's some nuance here because there's little consistency in what companies consider to be in Engineering's functional scope—for example, Security might or might not be included.

Partner with your closest cross-functional partners, particularly Product and Sales (or whatever the leading go-to-market function is within your company), to establish a high-level quarter or half roadmap.

> This is the cross-functional agreement on the scope and timing of work to be done.

Each of these phases depends on the preceding phase as an input, so they must be done in sequence. Sometimes as you proceed to later phases you'll learn something that requires changing a previous phase, which is entirely expected. Get comfortable with the reality that planning is dynamic!

Planning in these three distinct phases reduces the number of dependencies in each step. Fewer dependencies lead to a simpler, more focused process. Although you might assume the opposite, I've found that artificially unbundling planning this way *increases* decision quality. For example, the financial plan focuses on revenue, expenses, and headcount, while holding the roadmap and functional portfolio allocation constant. This means that you can focus on getting the financial plan reasonably correct without getting into a debate about the specific product releases. Once you start down the path of juggling the financial plan's details with the product releases and the mix of functional and cross-functional priorities, things get more complex but are rarely more accurate.

These constraints create better decisions because constraints drive innovation; removing constraints drives overly simplistic magical thinking such as spreadsheets that show tripling Engineering this quarter will triple Product velocity. Planning with sequenced constraints feels unnatural at first, but it's worth the initial discomfort.

I'm not arguing that you should *never* change your financial plan or *never* change your roadmap outside of this sequence. If you complete your planning roadmap and still believe that a headcount adjustment is necessary, it's reasonable to have that discussion. However, I am a strong believer in sequencing these discussions rather than having them in tandem to allow more focused, rigorous decisions.

From here, we're going to drill into each of the three phases, covering both the general approach and the particular challenges that come up when pristine theory meets messy reality.

Phase 1: Establishing Your Financial Plan

In your non-executive career as an engineer at a private company, you may never see your company's financial plan. If you do see it, you may only get a brief flash without the time to dig into the details as discussed in Appendix C, "Reading a Profit & Loss Statement". That makes it easy to get surprised in executive roles when you finally realize that your company's financial plan is the foundation of all company planning.

Figure 4-1 shows an example of a financial plan, with targets for every business line's revenue, and each function's expenses within that business line. These are dense documents that contain a great deal of data.

	Q1	Q2	Q3	Q4	Year	Last year	YoY%
Revenue							
Consumer	625	687	755	830	2897	2012	31%
Self-service	318	349	383	421	1471	812	45%
Enterprise	52	57	62	68	239	12	95%
Total revenue	995	1093	1200	1319	4607	2936	38%
Cost of revenue							
Cost of consumer	133	146	160	176	615	452	27%
Cost of self-service	162	178	195	214	749	629	16%
Cost of enterprise	96	105	115	126	442	18	96%
Total cost of revenue	391	429	470	516	1806	1099	39%
Gross profit	604	664	730	803	2801	1737	38%
Operating expenses							
Sales and marketing	479	526	578	635	2218	1521	31%
Research and development	212	233	256	281	256	782	20%
General and administrative	90	99	108	118	108	310	25%
Total operating expenses	781	858	942	1034	942	2613	28%
Net loss	-177	-194	-212	-231	-212	-876	-8%

Figure 4-1. Sample financial plan

For example, Figure 4-1 shows the Consumer business line is growing 31% year-over-year, with only $133 million in expenses on $625 million of revenue in Q1. Conversely, the Enterprise business line is growing much faster at 95% year-over-year growth but has $96 million in expenses for only $52 million of revenue in Q1. Referring only to those two rows, you can have a very interesting set of discussions about the two business lines.

The executive team will need to pull together an updated financial plan every year, which comes down to generating three specific documents:

- A profit & loss statement showing revenue and costs, broken down by business line and function.
- A budget showing more detailed expenses by function, including vendors, contractors, and headcount.
- A headcount plan showing the specific roles in the organization, with an emphasis on the new roles to be hired.

With these three documents, there are enough constraints to begin the rest of your planning process. That's not to say that these documents are easy to pull together; they can be rather difficult.

The good news, from an Engineering perspective, is that you're a stakeholder in creating the financial plan rather than being directly responsible for it. Every Finance leader will have their own variation on the details but expect the executive team and business line leaders to iterate on a series of proposals until you finish with something that no one's quite happy with but seems plausible.

This process is significantly different depending on your company's stage (e.g., Series B versus Series G), and whether you're public or private. Look to peer companies for their processes rather than emulating much larger companies you've worked at previously. But keep in mind that the accuracy of these plans is inherently low because they depend on projecting financial outcomes without having a defined product roadmap or knowing what chaos the year may bring. Push a bit on the details, but don't exhaust yourself.

How to Capitalize Engineering Costs

Capitalizing software costs is a financial concern, not an Engineering one, driven by the Finance team's obligations to follow the Generally Accepted Accounting Principles (GAAP). Generally, the Finance team's auditors won't certify their annual financial audits unless they've followed GAAP, and part of following GAAP is deciding whether to capitalize or expense each incurred cost.

The general guidance on capitalizing software engineering costs is straightforward: development of new software capabilities should be capitalized, and everything else should be expensed. In practice, there's a great deal of flexibility in deciding what is or isn't a new capability. This is the same challenge as agreeing on whether a given task is really a new feature and bug fix: most changes can be reasonably argued either way.

A surprising number of Engineering capitalization discussions don't start with a clear articulation of what each side needs, which makes it very challenging to agree on a good approach. My experience is that there are only three core criteria that your approach to capitalization must address to keep everyone involved happy:

- It must be easy to explain and justify to an auditor.
- It must be available in a timely manner for Finance's financial reporting.
- It must minimize both the Finance and Engineering teams' ongoing efforts.

Before deciding on your approach to capitalization, make sure Engineering and Finance agree on their needs! Although you can solve this problem in many different ways, most organizations end up going with one of three approaches:

Ticket-based

Track the capitalization status and time for each ticket. Establish a default for tickets without a capitalization status (likely that it will be expensed), and then sum the hours of capitalizable work across tickets for each engineer, multiplied by hourly cost, to determine capitalizable engineering costs.

Project-based

Track hours for each project and determine a capitalization weighting for each project (80% capitalized, 0% capitalized, etc.) based on the nature of the work. Finance then combines this with a synthetic hourly Engineering cost based on the average team or organization hourly rate.

Role-based

Set a fixed percentage of time for each role. For example, 80% of product engineer time should be capitalized; 0% of infrastructure engineering time should be capitalized; and so on.

The first two approaches, ticket- and project-based, are particularly common. Engineering teams tend to prefer the role-based model, but many Finance teams and auditors will view this approach with skepticism, so it's less common than you might imagine, although some companies do indeed practice it. That said, it is common to see a hybrid ticket or project-plus-role-based approach: relying on tickets to track capitalizable work on product engineering teams but assuming other engineering teams have no capitalizable work.

THE REASONING BEHIND ENGINEERING'S ROLE IN THE FINANCIAL PLAN

When you switch from the executive team perspective to the engineering executive perspective, the problem gets a bit simpler. Engineering is rarely directly accountable for revenue (although almost always indirectly accountable to either Product or Sales for that revenue), so your primary contribution to the financial plan is managing Engineering's expenses.

I recommend segmenting Engineering expenses by business line, into three specific buckets:

- Headcount expenses within Engineering
- Production operating costs (most cloud costs, vendor costs related to production, etc.)
- Development costs (vendor and hosting costs related to CI/CD, running test, development environments, etc.)

Within those segments, you should spend minimal time on business lines where revenue is accelerating faster than expenses (the business' quality is already improving), and a great deal of time on all other business lines.

For any business line where expenses accelerate faster than revenue, you and the entire executive team should have a clear, documented hypothesis for when and how you'll reverse the setup. It's not always a problem—it's expected for new business lines to spend a while in that phase—but it's essential that the overall executive team is marching to the same orders on each given business line.

WHY SHOULD FINANCIAL PLANNING BE AN ANNUAL PROCESS?

Your company's financial plan is the foundational constraint for every function. Most companies adjust their plan over the course of the year, which is appropriate, but I'd argue that you'll be more effective as an executive team and as an Engineering function if you operate as if the plan is fixed.

There are three core reasons for this:

- Adjusting your financial plan too frequently makes it impossible to grade execution, because your targets will keep moving.
- Making significant adjustments to your financial plan is a planning-intensive activity that requires a great deal of time across many functions. As such, changing the plan creates a great deal of churn, and often requires the reworking of other planning phases.
- Like all good constraints, if you make the plan durable, then it will focus teams on executing effectively within it. If you make it flexible, then teams will instead focus on moving the constraint (e.g., asking for more headcount). The former is much preferable to the latter.

Certainly, if you must, then you must, but it's much easier to run an effective company with a stable financial plan.

ATTRIBUTING COSTS TO BUSINESS UNITS

Early companies have a single business, which makes things simple. All Engineering costs are tied directly to running that one business. As companies grow, they'll eventually expand to multiple lines of business, where things get a bit trickier.

Even the simplest attributions get messy as you dig in. For example, consider a company like Figma, which has one core product but likely segments its business into two business units: Enterprise and everything else. The core product is the same in both cases, but many Enterprise features aren't valuable to non-Enterprise buyers. In that case, it's easy to attribute Enterprise-focused product engineers to the Enterprise business unit, but less clear how you should attribute product engineers building the core product. Attribute according to revenue? Attribute all costs to the non-Enterprise segment and show artificially good margins in Enterprise? Do something else?

The answer to all of these questions is always, "It depends." My experience is that there's relatively little precision on this topic. Focus on finding a defensible methodology that Finance is comfortable with and try to avoid reopening the dispute too frequently.

WHY CAN FINANCIAL PLANNING BE SO CONTENTIOUS?

Financial planning is not inherently contentious in well-run management teams. However, when executives are focused on their function rather than the executive team's shared success, allocating expenses is a zero-sum game. Combined with the tendency for struggling executives to point at an insufficient budget as the rationale behind their underperformance, it's easy to end up with either broken executive relationships or out of control spending.

If your financial planning process is contentious, I'd recommend pushing a bit on the topic with your CEO. It's most likely a sign of an executive team struggling to partner, or a particular struggling executive, rather than an issue that you can solve directly.

SHOULD ENGINEERING HEADCOUNT GROWTH LIMIT COMPANY HEADCOUNT GROWTH?

In general, I do believe that most companies should constrain overall headcount growth based on their growth rate for Engineering. This creates accountability to operate an efficient company, even when it's painful to do so. It also avoids the difficult scenario in which other organizations scale more rapidly than

Engineering and end up starving Engineering's bandwidth to make progress on the company's highest priorities.

There are, of course, always exceptions in the details. Uber did a good job of rapidly scaling city-specific operational teams with flexible tools that empowered them without letting them overload Engineering with requests. Uber may well have lost significant market share during their chaotic stretch of hypergrowth if they had constrained operational growth on the Engineering headcount.

INFORMING ORGANIZATIONAL STRUCTURE

Whenever you request additional headcount, you should have a documented organizational design to incorporate that headcount. The easiest way to do this is using some rough organizational math:

1. Divide your total headcount into teams of eight. Each of these should have a manager and a mission.
2. Group those teams into clusters of four to six. Each cluster should have an experienced manager and a focus area (e.g., Consumer Products, Enterprise Products, or Infrastructure).
3. Continue recursively grouping until you get down to five to seven groups, which will be your direct reports. In a company of ~40 engineers, you only need to form one tier of groups, but a company of ~200 engineers will require multiple tiers.

Although this will feel a bit superficial, I wouldn't recommend thinking about this in more detail during the financial planning process. There's a final phase of real organizational design that has to account for the strengths and experiences of the individuals at hand, but that degree of specificity isn't helpful during the financial planning process.

ALIGNING THE HIRING PLAN AND RECRUITING BANDWIDTH

The last financial planning topic I'll mention is that it's common for executive teams and leaders at the next layer to spend time arguing about headcount that's unlikely to get filled. I find it extremely helpful to compare historical recruiting capacity against the current hiring plan. If it's unlikely that you'll be able to make the planned hires based on the historical numbers, then I wouldn't spend too much time arguing about it.

This is particularly common in hyper-growth companies, where the executive team is not overly focused on R&D expenses and may greenlight most

headcount requests. In practice, that scenario leads to teams within R&D acquiring headcount by capturing recruiter bandwidth (e.g., getting specific recruiters assigned to their team's hiring pipeline) rather than through headcount approval. If you do find yourself in this situation, my recommendation is to align yourself with Recruiting by being a strong hiring partner: recruiters are graded on their hiring outcomes, and everyone likes being successful.

Renewing Vendor Contracts

In an established company, you'll have a procurement team to partner with on vendor negotiations. Lean on them to guide your negotiations. It can be trickier at small companies. You can walk into work, have someone mention a critical contract expires in two weeks, and find yourself unexpectedly learning to negotiate with vendors. If that's you, here are some tips.

Vendors generally only value a few things:

- Money, preferably guaranteed
- A long-term commitment, preferably exclusive
- Help meeting their internal goals, including hitting their quarterly quotas and buying higher-margin services (ever wonder why they'd rather reduce the total price than remove that training fee?)
- Sometimes timing of payments (for example, a small company with a capital-intensive business will really value upfront payment over on-demand payment)
- Sometimes co-marketing (if you'd help appeal to their future customers or investors)

On the other hand, what *you* generally want from vendors as you renegotiate deals is:

- A volume-based discount so that as you spend more, their rates decrease. You should always pay a lower rate as your spend increases.

- A commitment length that aligns with your potential timeline to consider other vendors. Generally, prefer one year unless you're getting something meaningful in return.
- Discounts based on service and delivery issues during the previous contract. If they had frequent outages, this should be reflected in the following contract to some extent.

The general approach I would recommend is:

1. Figure out if anyone is still using the vendor's or competitors' products. Often, they aren't.
2. Look for major issues that need to be addressed in renewal, such as API instability or rate limits.
3. Before you go into your first discussion with the vendor, talk through the negotiation game plan with whoever is going to sign the contract. Always have a strategy! In particular, figure out the volume discount and any poor-service discounts you want.
4. Go into the first negotiation focused on getting good numbers. Emphasize that *you* are taking the lead on the contract. (This can be a lie.)
5. Once you get written numbers back from the vendor, shift the negotiation to the contract signer and say that regrettably they've taken over the negotiation from you. (This can be a technicality.)
6. Have the contract signer push the vendor a bit more on pricing.
7. Sign the contract and move on to your next problem!

   This won't get you the very best pricing by any means, but it's reliable, doesn't take much time, and will save you a chunk of money. As you do more of these, you'll get more nuanced on your approach and will find ways to drive deeper discounts, but don't worry about that too much on your first contract negotiation: get a reasonable win, and then move on.

Phase 2: Determining Your Functional Portfolio Allocation

The second phase in your planning is your functional portfolio allocation. Remember from Chapter 3, that your functional portfolio allocation is part of your Engineering strategy and describes how you allocate Engineering resources against your current priorities. This functional portfolio allocation applies to your entire Engineering budget across vendors, contractors, and full-time headcount. This allocation directly impacts planning.

The fundamental question to answer in determining your functional portfolio allocation is how much Engineering capacity do you want to focus on stakeholder requests versus investing into internal priorities each month over the next year? This is just a series of percentages—for example, 63% of Engineering time in June will be focused on cross-functional projects, 75% in July, and 60% in August. However, deciding on these numbers can be difficult, and they have a significant impact on the company roadmapping.

Figure 4-2 shows one potential answer to that question, showing a fixed Infrastructure investment, expanding Developer Experience investment, and a temporary inwards shift in Product Engineering.

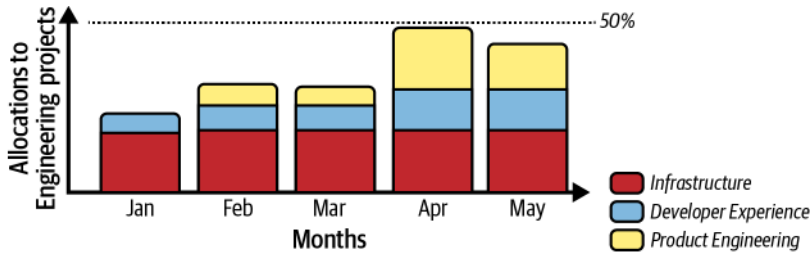


Figure 4-2. Functional allocation to Engineering priorities

Despite being a simple percentage, determining the correct percentage allocation is inevitably a bit trickier. The approach I recommend is:

1. Once a year or so, review your full set of Engineering investments, their impact, and the potential investments that you haven't made yet. Build the list of potential impacts from broad sources: your developer productivity survey, explicit brainstorming, and so on.
2. Update this list on a real-time basis as work completes and new ideas are suggested.

3. As the list is updated, revise your target steady-state allocation to functional priorities. Concretely, this is your investment into Platform and Infrastructure Engineering teams that exclusively work on functional priorities.
4. Aim to solve all functional priorities within your steady-state allocation, but be willing to discuss whether teams that generally do cross-functional work, such as Product Engineering, should temporarily assist for particularly high-impact or context-dependent projects.
5. Invest leadership time, including your own, into spot fixing the large allocations that are not obviously returning much impact. These are the places where you can learn the most about your organization, and also make the most meaningful adjustments.

If you're debating how much energy to invest into this process, remember that it doesn't need to be perfect; it needs to be useful. If your cross-functional partners are happy and engineers are happy, then you can get away with something very lightweight.

WHY DO WE NEED A FUNCTIONAL PORTFOLIO ALLOCATION?

If setting the company's budget and roadmapping is a joint exercise, it's worth asking why setting the functional portfolio allocation isn't also done by the executive team as a whole. The simple answer is that functional planning is best done by the responsible executive and team. Functional allocation relies so deeply on function-specific expertise that doing it in a larger, cross-functional group leads to worse outcomes.

That's not to say that functional leaders should allocate in isolation. I would strongly recommend doing it in partnership with your Engineering leadership team and other senior members of Engineering (as described in Chapter 3) and then verifying your proposal with peer executives. However, including peer executives too deeply is unlikely to help them or you: something has gone very wrong if you're trying to prioritize your CFO's opinion about monorepos over your staff engineers' perspectives.

Over time, address the root of this problem by slimming your allocation to functional priorities by converting them into cross-functional priorities. Things like compliance, security, reliability, and so on should really be fundamental company work, not invisible Engineering functional work. Bringing Engineering

work out of functional allocation is much more effective than trying to bring non-Engineering executives in.

KEEP THE ALLOCATION FAIRLY STEADY

The most common allocation challenge for executives is adjusting their allocation too frequently. You should prefer continuity and narrow changes over continually pursuing a theoretically ideal allocation. Changing your allocation feels like progress, but each change is fairly disruptive, so there's a significant penalty to making frequent changes. As such, I recommend a strong bias toward the resource allocation status quo, even when your current allocation is slightly suboptimal.

Another reason to make infrequent allocation changes is that teams can become overly focused on competing for allocation, which results in a zero-sum competition with their peers. That's a bad place to be culturally and distracts from the more valuable opportunity for creative problem solving, which has potentially infinite returns without promoting cross-team competition.

BE MINDFUL OF ALLOCATION GRANULARITY

There's an inherent trade-off between allocating at large (e.g., Infrastructure and Product) and small (e.g., Frontend Platform team and North America Growth team) granularities. The larger the granularity that you allocate, the more you empower your team to lead their teams. For example, if you allocate to Infrastructure Engineering, then Infrastructure's leader is empowered to make shifts within Infrastructure. If you make explicit allocations to teams within Infrastructure, then Infrastructure's leader would be obligated to work with you on changing their own allocations.

DON'T OVER-INDEX ON EARLY RESULTS

The most frequent concrete error I see in Engineering portfolio allocation is overestimating impact, or underestimating cost, based on early results. A couple of illustrative examples:

Example 1

> Two engineers work on improving build performance and speed up build times by 50% in two months. They argue that this shows you should theoretically invest 50% of Engineering capacity into developer productivity, and if not 50%, you should at least triple the size of the team. It's easy to invest more here, but they may (or may not) have already captured the early returns, meaning future results will be much lower.

Example 2

> Two engineers are working to migrate all frontend code into a shared frontend monorepo. After three months of work, you have a proof of concept, four annoyed frontend engineering teams, and no metrics showing an improvement. It's easy to cancel this project, but it may (or may not) be about to deliver significant gains, meaning future results will be much higher.

To prevent these mistakes, I recommend establishing a fixed investment into projects until there is at least one inflection in their impact curve. For example, keep the two engineers working on build performance until you see their impact shift downward, and only then estimate the correct long-term level of investment. If this feels too slow, then spend some time designing projects to show inflection earlier.

Phase 3: Agreeing on the Roadmap

There are many areas where I've found a definitive solution that I strongly believe in. Roadmapping is not one of them. You can get there with a four-column spreadsheet—for example, showing project, opportunity, implementation cost, and the owning team. There are innumerable other ways to create a roadmap that can work as well, and it's rarely the format that causes roadmapping to fail. (I recommend using whatever is already being proposed rather than spending time arguing about the format.)

Instead, roadmapping fails because of:

- Coupling roadmapping with changes in budget or functional portfolio allocation
- Tension between Product, Engineering, and their stakeholders
- Roadmapping a mix of concrete and unscoped projects
- Disempowering teams by planning too deeply

We've already spent time on the coupling problem, so let's dig a bit into the others.

ROADMAPPING WITH DISCONNECTED PLANNERS

Effective roadmapping requires Product, Engineering, and Design to work together collaboratively. While one function often leads planning, the other two should be active partners. Whenever that isn't the case, you will generate a roadmap that seems reasonable but fails to account for the real underlying topography of the planned work.

Similarly, I've often seen roadmaps where Product, Engineering, and Design are closely aligned with each other but the proposed work doesn't account for stakeholder requests (generally from Sales or Marketing). In these cases, a properly scoped roadmap will come together, but it won't generally be a set of work that optimally solves the company's challenges.

You can quickly determine if this is happening by checking with folks in other functions to see if they *generally* agree with the proposed plans. If not, pull the different functional planners into a room and hash it out. The frequent mistake I see here is executives trying to solve this through a series of one-on-one discussions. Debugging the situation one-on-one works well, but resolving it requires an open group discussion. I've had particularly good success with structured group discussions where each person has one to two minutes to share their perspective, without interruption, before the group starts the discussion.

ROADMAPPING CONCRETE AND UNSCOPED WORK

In Kellan Elliott-McCrea's excellent post "How to plan?", he makes an important observation that planning processes often proactively invite new ideas:

> *The implicit or explicit ask of Planning is often, "Tell us about all your exciting new ideas. We need your creativity to achieve our goals. Swing for the fences! Throw big rocks!" This is a mistake.*

The challenge is that new ideas are unscoped and unproven, and you end up comparing the raw potential of an unscoped idea against the proven potential of an existing idea. How you discount the unproven idea is usually a reflection of the executive team's psychology rather than its actual potential, which makes planning feel capricious. It also means, in the case that new ideas are quickly disproven, that your planning process will generate short-lived plans.

Two effective techniques for avoiding this scenario are:

- Agree on an allocation between scoped and unscoped initiatives, and then prioritize like-against-like within those allocations. For example, you might say that 20% of Product Engineering time should go against validating new projects. Then you can debate which unvalidated projects should staff that 20%, followed by a separate discussion around investing the 80% of time in validated projects.
- Have a long-running allocation toward validating projects, say 10% of Product Engineering time, which provides enough validation that you can reasonably prioritize those projects against existing projects.

Even if you're not able to formally adopt either of these approaches, you can often informally steer the discussion toward separating scoped and unscoped projects.

ROADMAPPING IN TOO MUCH DETAIL

The final roadmapping challenge I'll note is that if you get too specific, you can accidentally disempower the team responsible for the actual work. Melissa Perri captures this idea eloquently in *Escaping the Build Trap* (O'Reilly), which describes how roadmapping that is focused too narrowly on project to-do items rather than on desired outcomes can constrain teams' thinking.

Sometimes new leaders will take this concern a bit too literally and argue that executives should not be allowed to discuss specific projects, because it strips their team's autonomy. That's a bad outcome for everyone because it turns the executive team into a pure resource allocation decider, which is a bit too abstract a way to run a business until it gets exceptionally large.

Instead, the goal should be to emphasize the target outcome first at a high level of confidence, and then discuss the specific project second with a lower degree of confidence. As long as they remain aligned with the target outcome, the team should feel empowered to change the project. Digging into the specific project will surface a much richer discussion around execution than any abstract discussion about goals.

Timeline for Planning Processes

Mapping the three phases of planning to a calendar usually looks like this:

- Annual budget should be prepared at the end of the prior year.
- Functional planning should occur on a rolling basis throughout the year.
- Quarterly planning should occur in the several weeks preceding each quarter (if you're in halves instead of quarters, you'd prepare in the weeks preceding each half).

This typical implementation is shown in Figure 4-3.

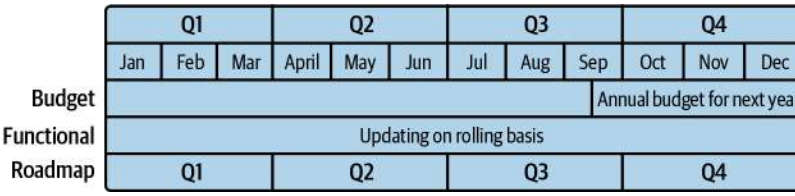


Figure 4-3. Example timeline for planning processes

The particular details here will reasonably vary across companies, and I have not found the details to matter too much one way or another. The biggest thing I recommend remaining vigilant against is something that Claire Hughes Johnson would often remark about when we began planning at Stripe: planning efforts always expand to fill the allocated time. If you set aside one week for planning, it will take one week. If you schedule one month for planning, it will take one month. Because planning is an infinite game, it's best to avoid spending too much time on any given finite step.

At some point in time, you may find yourself in a planning process run by someone who is unwilling to independently solve for the budget, functional priorities, and roadmaps. Similarly, you may find yourself working with a planning process that insists on solving every company problem, even though it's clear that it's solving all those problems poorly.

As an executive, you should work with the owner to improve the planning process, but you may also need to run a shadow planning

process within your function. This is straightforward to do for both the budget and the functional resource allocation. For example, you can usually establish a conservative annual headcount plan for Engineering and run Engineering toward that headcount plan, even if other functions jockey for more headcount over the course of the year.

On the other hand, it's pretty well impossible to run a shadow roadmapping process, even if the roadmapping process is difficult to work with. I've never seen a shadow roadmapping process work well except in the case of teams with exclusively Engineering customers such as Platform Engineering, Developer Productivity, or Infrastructure Engineering.

Pitfalls to Avoid

Even talented executive teams often run planning processes that go awry. The causes are a mix of incompatible goals, subtle defections from team goals to individual goals, and a long list of small, tactical errors with outsized consequences. To help you in diagnosing your planning challenges, I've pulled together the most frequent planning challenges I've encountered as well as the symptoms that you can use to identify which challenge is impacting your planning process.

PLANNING AS TICKING CHECKBOXES

Signs that you've delegated planning to someone who is process-oriented rather than outcome-oriented, or someone who is unable to push back on others demanding process-oriented changes, are:

Planning is a ritual rather than part of doing work.
: Teams put together dozens of planning artifacts to support the planning process, and the executives celebrate the depth of work necessary. After planning ends, the plans are rarely, if ever, referenced again until the next planning cycle begins. This incentivizes individuals to focus on optically valuable work rather than genuinely valuable work.

Planning is focused on format rather than quality.
: Executives feel obligated to provide input on plans to show that they value the planning process. If they don't have valuable feedback, they will still find feedback, often related to correctly following the planning documents' format or planning process. This distracts from assessing the quality of the planning decisions themselves.

An effective planning process must have an empowered executive sponsor who can keep it focused on its core goals, as well as a planning implementor who is at least as passionate about generating useful plans as running a clear process. The executive provides direction and makes trade-offs, but it's the planning implementor who designs the templates, finalizes the schedule, manages exceptions, checks to make sure initial drafts are useful, and generally keeps the process on track.

PLANNING AS INEFFICIENT RESOURCE ALLOCATOR

Signs that the executive team is optimizing for their function, rather than working together to optimize for the company's overall outcomes, are:

Planning creates a budget, then ignores it.

Many executive teams will establish a resource allocation across functions in their budgeting process but will then make headcount requests that violate that allocation. This leads to requests that are assessed by secondary factors, often individual's interests or demands, rather than by their alignment with company goals.

Planning rewards the least efficient organizations.

Often leaders will sandbag their organization's capabilities, arguing that they need significantly more headcount to even continue operating at their current level. This culminates in a business where most resources are directed toward your least efficient leaders and organizations.

A variant of this pattern is one where executives imply they can't be accountable for their function's output unless their full headcount request is met. This, combined with consistently high headcount requests, results in a toxic cycle of executives missing their obligations and claiming they can't be held accountable.

Planning treats headcount as a universal cure.

In headcount-oriented executive teams, it's easy for planning to become focused on rationalizing headcount requests rather than deciding on the most important work to be done. It's clear this is happening when prioritization and planning discussions assume velocity is fixed and that work must happen, and instead anchor on headcount. This penalizes creative leaders who understand how their function works and whose knowledge of how the function operates makes them appear less strategic than headcount-oriented peers.

An executive team can put social pressure on its members to optimize for the greater good, but ultimately only the CEO can hold executives accountable for their behavior. It is valuable to politely raise these issues, but only the CEO can address them. Pushing hard to resolve them will only backfire, as discussed in Chapter 18.

PLANNING AS REWARDING SHINY PROJECTS

Signs that your planning process is more focused on energizing the executive team than solving the inherent resource allocation and functional alignment problems at hand are:

Planning is anchored on work the executive team finds most interesting.

Certain projects will always be more energizing for any given executive team to discuss. For example, most executive teams are excited to discuss sales numbers or new product development, but fewer are thrilled to discuss compliance. If planning processes are driven by work that executives find fun, then they will frequently lead to poor planning outcomes.

Planning only accounts for cross-functional requests.

Many planning processes are focused on meeting cross-functional commitments rather than planning the entirety of work to be done. It's reasonable for roadmapping to only deeply consider cross-functional requests, but it's essential that the functional allocation is also being planned so that the executive team can discuss critical work that happens to be considered a single function's responsibility. For example, some companies view customer satisfaction, security, compliance, and stability as the responsibility of a single function.

Creating impactful planning outcomes depends on solving the business and functional problems at hand. Energizing the executive team is important, but distracting planning toward this goal comes with an unreasonably high cost.

PLANNING AS DIMINISHING OWNERSHIP

Signs that the executive team is approaching planning in a way that is reducing autonomy and ownership within your broader team are:

Planning is narrowly focused on project prioritization.

An effective planning process serves as guidance for teams within the company to accomplish their work. Executive teams sometimes focus too

narrowly on prioritizing specific projects rather than on the necessary outcomes or areas of investment. This often results in the teams with the most context, those implementing the project themselves, being unable to tune their approach to be more effective. This leads to disengaged teams and executives who are frustrated by the lack of urgency in the company's execution. It's more effective for executives to prioritize outcomes, which leave sufficient flexibility in their execution, than to steer projects toward those outcomes that heavily constrain non-executive leadership.

Planning generates new projects.

In some planning processes, it's the only time that some executives think about a given area. For example, your Chief Finance Officer may not spend much time thinking about the Customer Success organization's roadmap outside of planning. This is not inherently a problem, but sometimes those distant executives will use planning as an opportunity to suggest significant new, unscoped work. This will almost always result in a delayed or failed planning process, as it leads to attempting to prioritize well-scoped work against unscoped work, which is very challenging.

Executives should prioritize coaching the broader team through planning. At times, you will identify a leader who is not able to plan in their area, and at times you may need to make top-down changes to the plan—but these should be exceptions, not the norm.

Summary

Working through this chapter, you've dealt with the executive team's annual budgeting process, maintaining Engineering's target functional allocation, and combining the two to establish a concrete roadmap. You've also worked through a number of smaller topics, like whether Engineering growth should be the limiter on your company's overall growth, considering the granularity within your functional allocation, and the trade-offs of running a shadow planning process.

As you take these ideas into your next planning process, my biggest hope is that you remember that planning is an infinite game, and there's always a bit of mess at every stage. The key thing is to continue improving bit by bit!

Find links to further reading and resources on *https://lethain.com/eeprimer-refs-4*.

Creating Useful Organizational Values

Uber's best known corporate value is probably *Super Pumped*, which, in addition to being a one-time company value is also the title of Mike Isaac's account of Uber and the subsequent television show. However, personally, the value I remember most from Uber is *Let Builders Build* I particularly remember the handful of engineers who would claim that any objection to their proposals was a cultural failure to let them build what they wanted. This friction around a well-intentioned value shows just how challenging it can be to use values to operate a company.

As an Engineering executive, you will likely be involved in creating your company's values, interpreting how those values apply to Engineering, and deciding whether Engineering has additional values of its own. In this chapter I will:

- Clarify the kinds of problems that values solve effectively
- Discuss whether organizations (such as Engineering) should have values distinct from company-wide values
- Propose the ideal time to establish values
- Provide a clear method for evaluating proposed values
- Describe the distinction between Engineering values and your Engineering strategy
- Share an approach for rolling values out
- End with a discussion of the values that I've personally found effective within Engineering

Stated values possess no magic for cultural transformation, but with a bit of care they can be useful for consolidating cultural change that you've already made. You will leave this chapter prepared to decide whether stated values would be a useful addition to your organization today and, if so, how to create them.

What Problems Do Values Solve?

Stated values make it clear how you want people to make decisions. If senior team members live and model the values, then those values will become an active, living force. If stated values don't align with behavior, then they'll become a dusty, forgotten relic.

Here are some scenarios representing problems that values solve:

- You're hiring more and more senior engineers and managers, who come with a significant amount of work experience at other companies. Documented values increase cohesion across the new and existing team. They also avoid the scenario in which new hires unknowingly practice their previous companies' values, causing a cultural rift between new and existing teams.
- You've made a significant cultural change over the past year to welcome ideas from across the team, rather than only accepting top-down ideas, and you want to formalize that change so it persists over time.
- Engineering generally does a good job of reusing existing coding patterns and services, but a few vocal engineers are advocating for teams setting their own patterns independently of the organization's existing ones. Formalizing the reuse of existing approaches, when possible, will prevent a prolonged conflict.
- You've acquired a very small company of five engineers to join your existing organization of 500 engineers. You've been clear in the acquisition process that you're looking for the new team to merge into the existing organization, and your documented values help them navigate that change successfully.

Values are extremely useful in all of these scenarios, but it's important to recognize they are absolutely not an effective way to *start* the underlying culture change that they describe. Rather, they celebrate change that's already in motion, and help make it permanent. (If this strikes you as controversial, think about how

it holds true in your personal life. Claiming that you value punctuality doesn't make it true; consistently arriving on time does.)

Should Engineering Organizations Have Values?

Before digging into crafting *useful* values, it's worth spending some time discussing whether Engineering should have values at all. Organization-specific values tend to be controversial. Even if you've never heard of them, most companies have a written set of values somewhere, and many executive teams will take it poorly if you introduce Engineering-specific values. Why not simply use the existing company-wide values instead?

The best case is that you find company values that can be extended to match your goals. You might take Amazon's leadership principle of *Frugality* and add some Engineering-specific nuance: in our pursuit of *Frugality*, we consider the full cost of building and maintaining a new service against the cost of paying a vendor, and we find using a vendor is usually more frugal than building the new service ourselves. Adding an interpretation to an existing value reinforces that company value, rather than detracting from it, and is an easy sell to the wider executive team.

Sometimes, even if you get very creative at extending the existing values, there simply isn't an existing value that fits. In that case, there are three paths forward: add a new company value, introduce Engineering values, or introduce Engineering leadership values (for Engineering managers, Staff-plus engineers, or ideally both).

Which will work best depends heavily on your company size, your executive team, and the sort of values you're hoping to add. Some values simply aren't as relevant outside of Engineering: a value around build-versus-buy makes less sense for organizations that don't have the build option, and might make more sense as an Engineering value. Other values might work well for an entire company, such as prioritizing internal hires over external ones.

If the value is applicable to the whole company, and there's excitement within the executive team to add another value, then I'd encourage you to start there. It's much more work to maintain values than to create them, and adding to the company values will allow you to share the maintenance across the executive team. If the value is not widely applicable, then you have the choice between adding either Engineering values or Engineering leadership values.

Nominally, Engineering leadership values only apply to Staff-plus engineers and Engineering managers. In practice, almost everyone aspires to be a leader,

and will model their behavior on leaders within the team, so leadership values tend to establish themselves as organizational values while side-stepping some of the tripwires that come with being explicit.

In addition to personally finding success with leadership values, there are other examples in the wild, notably Amazon's Principal Engineering Community Tenets. (Note that these tenets apply *really well* to anyone in Engineering, regardless of their title.)

What Makes a Value Useful?

Companies sure do have a lot of stated values, but I am going to argue here that a significant number of those stated values are simply not useful. Often someone in the value-statement-creation process gets caught up in the dream of what the company *could* be, rather than what it *is*, and the values detach from reality. This turns a useful project into something that is at best useless, if not actively misleading. To avoid creating useless values of your own, walk through my short definition of what makes a value useful: a useful value is reversible, applicable, and honest.

Reversible: It can be rewritten to have a different or opposite perspective without being nonsensical.

Reversibility is a precondition to the next two values, applicability and honesty, but it's also meaningful in its own right. Effective values guide behavior, and it's only practical to guide behavior when there are multiple, viable approaches. Irreversible values invite lip-service agreement rather than active participation.

Example: Uber's value of *Make Magic* emphasized a willingness to delay product releases that were merely serviceable until they had a spark of delight. Many companies practice the reverse value of *Ship early and often*. Any given successful company might take either approach.

Counterexample: Amazon's *Are Right, A Lot*. How do you reverse that into a reasonable value? Something like "Are right when it matters" initially sounds like the reverse, but on deeper inspection probably means roughly the same thing. A more extreme reverse like *Are Wrong, A Lot* wouldn't make much sense unless you extend the value significantly to something like, *Quickly Learn From Mistakes*, which, again, is conceptually very similar to *Are Right, A Lot*.

Applicable: It can be used to navigate complex, real scenarios, particularly when making trade-offs.

Many values are written down and forgotten. To keep a value alive and useful, it needs to be used frequently and visibly by the team. A value is applicable when it contributes to planning sessions, performance reviews, and hiring decisions.

Example: Stripe's value of *Seek Feedback* clarifies the expectation that you should be actively seeking feedback on your work rather than working in isolation. If you've completed a tech spec without gathering input, then you know that at Stripe you're not actually done: you need to seek feedback before finishing the work.

Counterexample: Uber's *Be Yourself*. It's unclear how you're supposed to apply this value to a practical scenario. Another counterexample is Stripe's *Be meticulous in your craft*. It's a beautiful aspirational value, but it's sufficiently broad enough that two reasonable people can't align on whether it applies to any given decision.

Honest: It accurately describes real behavior.

A touch of aspiration is OK, but useful values should explain how effective employees navigate the organization as it exists today. If it's too aspirational, values become a trap for your best-intentioned employees. More cynical employees will ignore them anyway, modeling their behavior on the actions of successful internal role models rather than your stated values. The only way for your entire team to operate using the same values is to describe behavior honestly. (If you want to change company behavior, change the behavior first, and document it second.)

Example: Uber's value of *Meritocracy and Toe-Stepping* has been criticized externally, but it was an accurate reflection of how Uber's leadership wanted culture to work. You went into a meeting with folks at Uber, and that's how they acted.

Counterexample: Amazon's *Strive to be the Earth's Best Employer*. There are many things uniquely good about working at Amazon, but few would argue that their frugality makes them the best employer.

As a test, let's discuss a few values that meet all three criteria:

- Amazon's *Dive Deep* asks leaders to engage with the details at hand. If they're uncertain about the right path forward, they're urged to dig in, rather than delegate. This is reversible: many successful companies instead delegate decisions down the hierarchy to whoever has the details rather than expecting leaders to drill in themselves. This is applicable: whenever you're not sure about a given decision, you should dive into the details before moving forward. This is honest: most folks I've worked with from Amazon have embodied this mentality and are willing to go deep into problems.
- Stripe's *Seek Feedback* asks folks to socialize plans and decisions before finalizing them. This is reversible: other companies would urge moving quickly by avoiding consensus building. As discussed earlier, this is applicable because you know any major decision needs to be written up and shared. Finally, it's also honest: this is how people behave at Stripe (when folks complain about Stripe, it's often that it works this way too much).
- Y Combinator's *Do Things That Don't Scale* asks companies to avoid premature optimization in their early efforts. This is reversible: many early companies want to emphasize doing quick work, but others focus on going slow to create something truly unique. This is applicable: whenever you're making a short-term versus long-term trade-off, you can apply this value. This is honest: most Y Combinator startups choose to test quickly and delay solving scale problems for later.

Each of these values is genuinely useful for deciding if a company is for you, and also for operating successfully within that company.

Some folks will argue that good values should be resistant to misuse. For example, Uber's *Let Builders Build* might be a bad value because it was misinterpreted by the Product Engineering manager. I don't personally worry too much about that. Self-interested individuals will *always* interpret things in unrealistic ways for their own benefit, and the solution is to hold them accountable for their poor behavior, particularly if they are role models like a Staff-plus engineer or engineering manager.

There are two particularly common categories of non-useful (or colloquially, useless) values that occur frequently enough that they're worth discussing directly: identity values and prioritization values.

Identity values are things like Amazon's *Are Right, A Lot*, Uber's *Champion Mindset* and Stripe's *Deliver Outstanding Results*. Identity values can be reversible and may be honest, but they are very rarely applicable. How often are you finishing up a project where you think to yourself, "I should focus on being right, rather than wrong, for this one."

Prioritization values include Uber's *Celebrate Cities* and almost every startup's variant of *Make Big Bold Bets*. These values often are honest but struggle to be either reversible or applicable. Uber's *Celebrate Cities* asks that some work is done to acknowledge the cities that Uber operates in, but it doesn't provide any context to appropriately size or prioritize that work. Most importantly, if you dig into any prioritization value, you will usually find a hidden identity value. For example, making big bets is really about the desire to be innovative and ambitious.

While you should try to avoid useless values, as long as they're honest, they tend to be inert rather than harmful so I wouldn't spend too much time fighting against them, particularly if you're working on values as a participant (e.g., for the company) rather than as the final decider (e.g., for Engineering).

How Are Engineering Values Distinct from a Technology Strategy?

There's some intersection between Engineering values and Engineering strategy. The best way to think of the relationship between values and a strategy (business, technology, or otherwise) is that a useful value generally can serve as a strategy's guiding principle. Not all guiding principles are values—for example, how you respond to a current market opportunity is unlikely to be a value—but most values are viable guiding principles.

For example, *Reuse common technologies unless you see 10x improvement* is a useful Engineering value, and could well be a guiding principle you use to address a specific set of circumstances. While values are usually presented in a way that's divorced from the circumstances that help form them, you can reverse-engineer the implicit circumstances that fed into any given value's creation.

By default, folks often view it as a failure if their organizational values and strategy have too much overlap ("it's redundant to say the same thing in two different places"), but I see them as tightly connected. Rather than a bad sign, I view overlapping organizational values and strategy as a sign of attentive, detail-oriented leadership.

When and How to Roll Out Values

It often feels like there's a playbook that encourages new executives to quickly publish new values, similar to the nervous desire to show value that culminates in new executives announcing brand new technology architectures shortly after joining. Don't do that. Wait. Wait at *least* six months. If that's too long, at minimum wait until you can evaluate whether a given value is honest or aspirational. You can absolutely test some values earlier in small groups, but if you're confident you *need* to roll out values quickly after joining a company, I'd push you to consider whether this is the right work or if you're retreating into comfortable work.

By focusing on honest values, you'll have fewer rollout challenges from folks rejecting values outright, but the rollout will still be an involved process. Establishing values should follow the general patterns of good process rollout: identify the opportunity, document potential options, involve stakeholders early to build buy-in, test before finalizing to avoid folks feeling trapped, and iterate until it's useful. It's easy to announce values, but much harder to introduce values that get used. Reduce that risk by including the wider team, listening, and iterating a few times to synthesize a shared creation rather than a top-down one.

Once you've completed the initial rollout, it's important to recognize that values are more like a garden than a building. If they're not part of your daily processes, they'll probably be forgotten. Therefore, I recommend you:

- Integrate them into your hiring process (including letting candidates opt-out if they don't like them).
- Explicitly talk about them in new-hire onboarding.
- Update your career ladder to require value-aligned behavior for promotions.
- Highlight culture-aligned accomplishments in 1:1s, team meetings, and elsewhere.

The one caveat to the standard process rollout is that if you end up writing company values rather than Engineering or engineering leadership values, then you will likely have quite a few peer executives or the CEO involved, and the reality is that executives are hard to hold to a process. This is mostly unavoidable, and you'll just have to roll with the chaos a bit. (This is one of the reasons I think it's worthwhile to consider crafting Engineering leadership values as opposed to engaging with the company values overall.)

Some Values I've Found Useful

Particularly successful companies export their stated values to the next generation of companies, which tends to make some values appear universal. Amazon's *Customer Obsession* is a great example: How many companies formed after Amazon include a value around customer centricity? Almost all of them! How many of those companies replicate Amazon's genuine focus on customers? Remarkably few.

The challenge with copying values is tied directly to the definition of useful values: reversible, applicable, and honest. A value's reversibility is universal, but whether the value is applicable to a company's situation and an honest reflection of the company's behavior varies greatly across companies. For example, while many companies pursue *Frugality*, only some practice it, and it requires an honest self-appraisal to determine if *Frugality* is a useful value for you.

Despite these inherent hazards of copying values, I do want to share some of the values that I've found particularly valuable in Engineering organizations that I've worked in. I'm not arguing you should adopt these specifically but consider the rationale behind them and whether they might *honestly* apply to your organization as well:

Create capacity (rather than capture it).

This value focuses leaders on creating new capacity from outside the company, rather than fighting internally for existing capacity. Creating capacity makes the first-team mindset possible, because it aligns incentives across managers who would otherwise be competing with one another for budget.

Reversible? Yes. Many organizations take a "fungible headcount" view on shifting teams and individuals, which encourages leaders to capture capacity to complete their goals.

Applicable? Yes. This is highly relevant in any prioritization or headcount-planning exercise.

Default to vendors unless it's our core competency.

You can also write this as: build-versus-buy decisions should consider the full implementation and maintenance costs. It is often more intellectually interesting to build commodity solutions than to use existing vendor solutions, but the cost of operating, maintaining, and extending those solutions is often much higher. Good build-versus-buy decisions look beyond the initial implementation cost and don't prioritize initial fun over long-term maintenance pain.

Reversible? Yes. For example, early Uber had a strong "Not Invented Here" culture, and rarely used external vendors.

Applicable? Yes. This is relevant to any platform or capability build-versus-buy decision.

Follow existing patterns unless there's a need for order-of-magnitude improvement.

Engineering organizations often get caught in a slow burning but endless conflict about when to introduce new programming languages and tooling. It's exceptionally valuable to have a clear answer to align folks on making these decisions. (See the discussion on this in Appendix E.)

Reversible? Yes. Many companies encourage introducing new technology or have an infinitely high bar to adopting a new technology stack.

Applicable? Yes. This is relevant whenever selecting the technology stack for a new project.

Optimize for the [whole, business unit, team].

Pick *one* of these options to help leaders understand how they should balance decisions between impacting their team and impacting other teams. It's also a good example of how values and architecture intersect: you can't *really* optimize only for your team if you're in a monolithic service without causing significant problems elsewhere.

Reversible? Yes. There are successful organizations that optimize for the team, ones that optimize for the company, and so on. As long as the approach is aligned with the technical architecture, all of these are viable approaches.

Applicable? Yes. This is useful whenever making trade-offs across the team and organization—for example, when selecting the technology stack for a new project.

Approach conflict with curiosity.

One of my foundational beliefs is that most professional conflict between reasonable people is driven by asymmetric information. If you approach conflict with curiosity, you can quickly learn the missing information and generally make the right decision without conflict.

Reversible? Yes, although admittedly on the weaker side. A given successful company might focus on resolving conflict via subject matter expertise, data, user feedback, and so on.

Applicable? Yes. This is useful in any scenario with conflict.

Summary

In this chapter, you have learned to evaluate values according to their reversibility, applicability, and honesty. You've used that approach to assess several well-known company values, and even the Engineering values that I have personally found valuable in my work. You now have the tools to decide whether to prioritize establishing Engineering values, company values, or if there's still foundational work to make those values honest before cementing them into formal documentation.

Find links to further reading and resources on *https://lethain.com/eeprimer-refs-5*.

Measuring Engineering Organizations

For the past several years, I've run a learning circle with Engineering executives. The most frequent topic is friction with their CEO. The second most frequent is what to measure when their CEO asks for a periodic report on Engineering metrics. Any discussion about measuring Engineering organizations quickly unearths strong opinions. Some of the strongly held perspectives that I've heard are:

- Tracking Agile story points is either essential or nearly criminal.
- Objectives and key results (OKRs) are done by every serious company, or rarely done at all.
- Engineering is just developer productivity, which you can track using deploy frequency, cycle time, and so on.
- Tracking incident frequency and impact is very valuable, but tracking incidents is also a harmful antipattern.
- It doesn't matter what you track, because no one will read it anyway; just make a pretty chart.
- Just measure something related; what really matters is starting to educate your stakeholders rather than the particular metric.

All of these perspectives have a grain of truth to them, but frustrations about measurement are easily overstated. There are many effective modes of engineering measurement, all of which will help you operate as an Engineering executive. You need to measure Engineering to build software more effectively,

and to hold your teams accountable for their work. You will also need to measure Engineering to collaborate cross-functionally, and to stay aligned with your CEO.

This chapter will walk through what you should measure for each of these scenarios to efficiently accomplish these varied goals. By the end, you will have a broad perspective on the options available for measuring Engineering, as well as the antipatterns to avoid as you begin measurement.

Tip

You can use my Engineering Metrics Update template to report on these measures.

Measuring for Yourself

When you enter a new executive role, your CEO will probably make an offhand remark that you will, "need to prepare an Engineering slide for the next board meeting in three weeks." Preparing that Engineering slide for the board is a very focusing moment but take a step back before concentrating on what the board wants.

Instead, start by measuring for *yourself*. What is the information that *you* need to operate the Engineering organization effectively? As you start to document your measurement goals, I recommend starting with these four distinct buckets.

MEASURE TO PLAN

What would help you align with cross-functional stakeholders to decide *what* to ship in a given quarter, half, or year? Critically, this is not about *how* you ship (whether you use Agile, or something else); rather, your goal is to support project selection and prioritization. You want a single place that shows the Business, Product, and Engineering impacts of work. Some organizations believe that planning is the Product team's responsibility, but only Engineering can represent the full set of projects that Engineering works on—Product is unlikely to roadmap your database migration. Start by tracking the number of shipped projects by team, and the impact of each of those projects.

MEASURE TO OPERATE

What do you need to know to be confident that your software and teams are operating effectively on a day-to-day basis? It can be helpful to think of these as measures of execution quality, and if they dip then that's cause to consider whether you should depart from your longer-term strategy to address the underlying issue at hand.

Some good places to start are: tracking the number of incidents (each connected to an incident review writeup), minutes of user-facing API and website downtime, latency of user-facing APIs and websites, engineering costs normalized against a core business metric (for example, cost to serve your most important API, calculated simply by dividing API requests per month by engineering spend for that month), user ratings of your product (e.g., App Store ratings), and a broad measure of whether your product's onboarding loop is completing (e.g., what percentage of users reaching your website successfully convert into customers).

MEASURE TO OPTIMIZE

What do you need to know to effectively invest time into improving Engineering productivity? If you look at Engineering as a system, how would you understand that system's feedback loops? It's worth noting that these measurements tend to be the basis for how engineers evaluate their experience working within your company.

Start with SPACE or its predecessor, *Accelerate*. Both frameworks identify measurable characteristics like delivery lead time and deployment frequency that correlate strongly with high productivity Engineering efforts. Instrumenting those measurements can be heavy for an initial lift, in which case, start with a developer productivity survey.

MEASURE TO INSPIRE AND ASPIRE

How can you concisely communicate Engineering's transformational impact on the business? How would you energize a potential hire, new cross-functional executive, or board member about Engineering's contribution to the business? Maintain a list of technical investments that turned something impossible into something so obvious that it isn't really worth mentioning. When you speak at company meetings, anchor on these kinds of generational improvements, and try to include at least one such improvement into every annual planning cycle.

As an example, imagine going from a scenario in which you have frequent user-facing instability due to one API endpoint's poor performance. If you replace that setup with an improved one in which each API endpoint is only allowed to take a certain share of overall capacity, then you will have hardened your API to instability cascading across endpoints.

A frequent rejoinder when you discuss measurement is whether you're measuring enough, because there's always more you could measure: database CPU load, pull request cycle time, and so on. The secret of good measurement

is *actually measuring something*. The number one measurement risk is *measuring nothing because you're trying to measure everything*. You can always measure more, and measuring more is useful, but measure across many small iterations and don't be afraid to measure something quick and easy today.

Measuring for Stakeholders

One reason to focus on quick measurements rather than perfect measurements is that there are a *lot* of folks within your company who want you to measure something (often to confirm or disprove their intuition about your current trendline). There's stuff that you find intrinsically valuable to measure, but there's also stuff that your CEO wants you to measure, that Finance is asking you for, to present at monthly status meetings, and so on. There are also requests from both strategic peer functions (those evaluated on their contribution to shared business goals, such as revenue growth or product adoption) and tactical peer functions (those evaluated primarily on their internal operations, such as a Customer Success function graded on its ticket response time and user satisfaction).

While there are quite a few asks, the good news is that the requests are common across companies and can usually be mapped into one of four categories.

MEASURE FOR YOUR CEO OR YOUR BOARD

Many experienced managers, likely including your CEO, manage through inspection (as we'll discuss in Chapter 17). This means they want you to measure something, set a concrete goal against it, and share updates on your progress against that goal. They don't have an engineering background to manage your technical decisions, so instead they'll treat your ability to execute against these goals as a proxy for the quality of your engineering leadership. If they dig into your measurements, they'll focus on Engineering's contribution to the business strategy itself, rather than the micro-optimizations of how Engineering works day-to-day.

These measures don't need to be novel, and your best bet is to reuse something you're already measuring for your own purposes. Most frequently, that means reusing your measures for planning or operations.

The biggest mistake to avoid is asking your CEO to measure Engineering through your *metrics to optimize*. Although intuitively you can agree that being more efficient is likely to make Engineering more impactful, most CEOs are not engineers and often won't be able to take that intuitive leap. Further, being highly efficient simply means that Engineering *could* be impactful, not that it actually *is* impactful!

MEASURE FOR FINANCE

Finance teams generally have three major questions for Engineering. First, how is actual headcount trending compared to budgeted headcount? Second, how are actual vendor costs trending compared to budgeted vendor costs? Third, which Engineering costs can be capitalized instead of expensed, and how do you justify that in the case of an audit?

You will generally have to meet Finance where they are within their budgeting processes, which vary by company but are generally unified within any given company. The capitalization-versus-expensing question is a nuanced one, which we covered in Chapter 4.

Although I've rarely had Finance ask for it, I find it particularly valuable to align with Finance on Engineering's investment thesis for allocating capacity across business priorities and business units. At some point, business units within your company will start arguing about how to attribute Engineering costs across different initiatives (because Engineering costs are high enough that allocating too many will make their new initiative incur a heavy loss rather than show promise). Having a clearly documented allocation will make that conversation much more constructive.

MEASURE FOR STRATEGIC PEER ORGANIZATIONS

Your best peer organizations are proactive partners in maximizing Engineering's business impact. Optimistically, this is because these functions are led by enlightened business visionaries. More practically, it's because their impact is constrained by Engineering's impact, so by optimizing Engineering's impact, they simultaneously optimize their own. It's common to find Product, Design and Sales functions that are well-positioned for strategic partnership with Engineering. You can usually align with strategic peer organizations using the measures mentioned in "Measure to Plan" on page 90.

MEASURE FOR TACTICAL PEER ORGANIZATIONS

While strategic peer organizations may agree to measure Engineering on the impact of its work, more tactical organizations will generally demand to measure Engineering based on more concrete outcomes. For example, a Customer Success organization may push Engineering to be measured by user ticket acceptance rate and resolution time. Legal may similarly want to measure you on legal ticket resolution time. Tactical organizations are not tactical because they lack the capacity for strategic thinking, but rather because their organization is graded in a purely tactical way. If they align with Engineering to maximize Engineering's

impact, it will often reduce progress against their targets (for example, time-to-first-response on incoming support tickets), causing them to be perceived as underachieving.

Tactical organizations are held accountable to specific, concrete numbers and likewise want to hold Engineering accountable to specific, concrete numbers of their own. Find something that you and the peer organization are able to agree upon, identify a cadence for discussing how effective that measure is at representing the real cross-organizational need, and iterate over time.

Measuring for stakeholders is often time-consuming, and at times can even feel like a bit of a waste. However, it ultimately saves time because it consolidates messy, recurring discussions into simpler, more predictable contracts about maintaining specific, measurable outcomes.

Sequencing Your Approach

The combined list of stuff to measure for yourself and others can be exceptionally long. Almost every executive has a much longer list of things they want to measure than they can resource measuring. You'll never actually measure everything you want. However, over the course of a year or two, you'll be able to instrument the most important, iterate on them, and build confidence in what they actually mean.

I tried to create a concrete list of exactly what you should measure first to avoid spreading yourself too thin, but it wasn't that useful. There's just too much context for a rote list. Instead, I offer three rules for sequencing your measurement asks:

1. Some things are difficult to measure, so only measure those if you'll actually incorporate that data into your decision making. If you're unlikely to change your approach or priorities based on the data, then measure something simple instead, even if it's just a proxy metric.
2. Some things are easy to measure, so be willing to measure those to build trust with your stakeholders, even if you don't find them exceptionally accurate. Most stakeholders will focus more on the intention and effort than the actual output. It's an affirmation that you're willing to be accountable for your work, not a testament to the thing being measured.
3. Whenever possible, only take on one new measurement task at a time. Measurement is challenging. Instrumentation can go awry. Data can be subtly wrong. Bringing on many new measures at once takes a lot of time

from you or the subset of your organization that has a talent for validating new data.

Work through measuring something from each of the broad categories (for your CEO, for yourself, and so on) to establish a baseline across those dimensions. Once you've completed that first pass, you can always return to take a second, third, and fourth pass on improving the areas where your initial measurements aren't quite working out. Even if you do an excellent job in your first pass, there will always be a new situational context that leads you toward a slightly different measurement approach.

That's fine! Measurement is an iterative process. Don't get caught up in following this plan exactly; take the parts that are most useful.

Antipatterns

Measurement is rife with antipatterns. The ways you can mess up measurement are truly innumerable, but there are some that happen so frequently that they're worth calling out:

Focusing on measurement when the bigger issue is a lack of trust.
: Sometimes you'll find yourself in a measurement loop. The CEO asks you to provide Engineering metrics, and you provide those metrics. Instead of seeming satisfied, the CEO asks for another, different set of metrics. Often the underlying issue here is a lack of trust. While metrics can support building trust, they are rarely enough on their own. Instead, push the CEO (or whoever remains frustrated despite your bringing more metrics) on their frustration, until you get to the bottom of the concern masquerading as a need for metrics.

Letting perfect be the enemy of good.
: Many measurement projects never make progress because the best options require data that you don't have. Instrumenting that data gets onto the roadmap but doesn't quite get prioritized. A year later, you're not measuring anything. Instead, push forward with anything reasonable, even if it's flawed.

Using optimization metrics to judge performance.
: It's easy to be tempted to use your optimization metrics to judge individual or team performance. For example, if a team is generating far fewer pull requests than other teams their size, it's easy to judge that team as less

productive. That may be true, but it might also mean they work in a more complex area of the codebase. Instead, evaluate teams based on their planning or operational metrics.

Measuring individuals rather than teams.
: Writing and operating software is a team activity, and while one engineer might be focused on writing code during this sprint, another might be focused on "being glue" to make it possible for the first engineer to focus. Looking at individual data can be useful for diagnostic purposes, but it's a poor tool for measuring performance. Instead, focus on measuring organization and team-level data. If something seems off, then consider digging into individual data to diagnose, but not to directly evaluate.

Worrying too much about measurements being misused.
: Many leaders are concerned that their CEO or board will misuse data. For example, they might throw a fit that many engineers are only releasing code twice a week, viewing it as a sign that the engineers are lazy. Recognize that these lines of discussion are very frustrating, but also that avoiding them doesn't fix anything! Instead, take the time to educate stakeholders who are interpreting data in unconstructive ways. Keep an open mind, as there is always something to learn, even if a particular interpretation is wrong.

Deciding alone rather than in community.
: I do recommend coming to measurement with a clear point of view on what you've seen work well, but that should absolutely be coupled with multiple rounds of feedback and iteration. Particularly when you're new to a company, it's easy to project your understanding of your last job onto the new one, which erodes trust. Instead, build trust by incorporating feedback from your team and peers.

If you've avoided these, you're not guaranteed to be on the right path, but you've increased your odds.

Building Confidence in Data

After you've rolled out your new Engineering measures, you may find that they aren't very useful. This may be because different parties interpret them in different ways ("Why is Finance so upset about database vendor

costs increasing when our total costs are still under budget?"). It may even be that the measures conflict with teams' experiences of working with the systems ("Why does the Scala team keep complaining about build latency when average build latency is going down?").

The reality is that measurement is not inherently valuable; it is only by repeatedly pushing on data that you get real value from it. A few suggestions for pushing on data to extract more value from it:

Review the data on a weekly cadence.
: Ensure that your review looks at how the data has changed over the last month, quarter, or year. Whenever possible, also establish a clear goal to compare against, as both setting and missing goals will help focus on the areas where you have the most to learn. Do this alone and document the questions you can't answer for future improvement.

Maintain a hypothesis for why the data changes.
: If the data changes in a way that violates your hypothesis ("cost-per-request should go down when requests-per-second go up, but in this case, the cost went up as requests-per-second went up—why is that?"), then dig in until you can fully explain why you were wrong. Use that new insight to refine both your hypothesis and your tracked metrics.

Avoid spending too much time alone with the data.
: Instead, bring others with different points of view to push on the data together. This will bring together many different hypotheses for how the data reacts and will allow the group to learn together.

Segment data to capture distinct experiences.
: Reliability and latency in Europe will be invisible—and potentially quite bad—if your data centers and the vast majority of your users are in the United States. Similarly, Scala, Python, and Go will have different underlying test and build strategies. Just because build times are decreasing on average doesn't mean Scala engineers are having a good build experience.

Discuss how the objective measurement does or does not correspond with the subjective experience of whatever the data represents.

If builds are getting faster, but don't feel faster to the engineers triggering them, then dig into that. What are you missing?

There are, of course, many more ways to push on data, but following these suggestions will set you on the right path toward understanding the details and limits of your data. If you don't spend this sort of time with your data, then it's easy to be data-driven and highly-informed but also come to the wrong conclusions. Some of the worst executive errors stem from relying too heavily on subtly flawed data, but this is avoidable with a bit of effort.

Summary

In this chapter, you learned different ways to measure Engineering; to support planning, operation, optimization; and to inspire the team. You've further learned to take elements from each of these techniques to measure on behalf of your CEO, peers, stakeholders, and yourself. While many Engineering leaders view measurement as a toilsome obligation, you now have a broad set of techniques to make measurement a fundamental part of effectively leading your organization.

Find links to further reading and resources on *https://lethain.com/eeprimer-refs-6*.

Participating in Mergers and Acquisitions

I managed the Engineering team at Digg as we ran out of money, and were eventually acquired. It was an eye-opening experience and I learned a great deal about the reality and the optics of selling a company, particularly one with no money and a shrinking user base. Humbling was just the beginning.

Since then, I've had the opportunity to evaluate a number of companies from the other side of the mergers and acquisitions (M&A) table. Most of those discussions didn't move forward, but a handful did, and every discussion was an education in Engineering's role in these messy processes. Acquisitions are filled with risk, with early discussions creating a great deal of anxiety for Finance, Legal, and People teams. Engineering has an interesting risk profile as well, and needs to evaluate the potential product value and integration cost of a potential acquisition.

However, and unlike many other functions, Engineering will be intimately involved after the acquisition closes, typically leading the integration and operation of the acquired offering. This incentivizes Engineering to be particularly careful in their assessment. While there are no prizes for diligently vetting acquisitions (often just a trail of annoyed colleagues), it's critical work. Thoughtfully integrating a trajectory-changing acquisition is one of the most impactful things you can do as an Engineering executive, and stopping a poorly thought-out acquisition from moving forward is similarly guaranteed to have an understated but massive impact on your company's trajectory.

In this chapter, we will work through:

- The complex, often misaligned, incentives of acquiring another company
- Developing a shared perspective—with a business strategy, acquisition thesis, and Engineering evaluation
- Topics you should cover in your Engineering evaluation
- Integration decisions you should make while implementing an acquisition
- Handling the flipside: being acquired

By the end, you'll have a structure for running the overall M&A process and a template for conducting Engineering's portion of that process. Further, I hope you'll agree with me that evaluating acquisitions is some of the most interesting work that Engineering executives do, and that managing Engineering's "no" in the M&A process is uniquely impactful work.

Complex Incentives

Even the simplest acquisitions suffer from messy, often conflicting, incentives. After one acquisition closed, I was responsible for integrating the team and product offering into our company. When the deal team stepped away—mission accomplished!—I quickly learned that the M&A deal team had handed off a bit of a mess for me. The engineers at the acquired company were told that they were being brought in to salvage our failing technology stack. Our engineers had been told that the acquired team knew they'd need to rewrite their stack into our standard stack, and that they were totally excited for it. This setup didn't create the smoothest start to our integration process.

This kind of miscommunication—some might call it a mild deceit—happens all the time in acquisitions. An important underlying truth of mergers and acquisitions is that the incentives can get very warped. Your M&A team may be evaluated by their ability to complete a certain number of acquisitions that contribute a certain amount of revenue to the business. That goal would align them toward moving forward with almost any revenue-contributing acquisition, even if it's an exceptionally painful integration. You as an Engineering executive might be very proud of your current technology stack and team culture, and consequently somewhat resistant to potential acquisition. Your company's founders or CEO will skew heavily toward acquisition-loving or acquisition-hating, mostly based on their previous experiences, which complicates the process as well. Similarly, the other company's founders or CEO will either be desperate for

an acquisition or totally resistant to it, and sometimes both over the course of one day: it's hard to give up a business you've built, even when it's quickly running out of cash.

Finally, there's the allure of the sunk cost fallacy after you've spent the better part of a year evaluating six or seven potential acquisitions without moving forward on any of them. At some point, the amount of effort the deal team has expended becomes hard to overlook, and you'll start biasing toward moving forward even if the full math doesn't check out.

Keeping these messy incentives in mind is fundamental to navigating acquisitions successfully.

Developing a Shared Perspective

Every acquisition, much like every new company, has a hundred reasons why it won't work. If you only look for reasons why an acquisition is a bad idea, you will always find quite a few. Conversely, if you only look at the potential good, every acquisition will look irrationally rosy. To navigate the tension between those two extremes, your company needs to develop three tools:

1. Business strategy: what is the desired role of M&A in your business?
2. Acquisition thesis: what are the assumptions that need to be true for this particular acquisition to fit into your business strategy?
3. Engineering evaluation: are you able to disprove any of those assumptions, or uncover other significant risks?

A surprising number of M&A processes do not include the first two tools, but you can see why that makes the third step an awkward formality: how do you disprove the unstated assumptions driving the potential acquisition? Your best option in those cases is to write the missing strategy and thesis yourself. Evaluating an acquisition without *each* of these tools—strategy, thesis, and evaluation—will end in tears.

Tip

If you want a detailed look into one company's approach to M&A, GitLab's acquisition process is documented online, and is quite good.

BUSINESS STRATEGY

From an M&A perspective, your business strategy is how you expect acquisitions to contribute to your company. It needs to be written down and it needs to answer a number of questions:

1. What are your business lines?
2. What are your revenue and cash-flow expectations for each business line?
3. How do you expect M&A to fit into those expectations?
4. Are you pursuing acquihires (for your team only), product acquisitions (with an emphasis on product capabilities and less focus on revenue and distribution), or business acquisitions (an entire business line with product, revenue, and distribution)?
5. What kinds and sizes of M&A would you consider?

Ideally, your CEO or the combined executive team is writing this, rather than you writing it alone from the Engineering perspective, but sometimes you'll end up writing the first draft out of necessity. If you do find yourself attempting to write this document, just write anything reasonable and share it with the executive team. To a certain extent, the worse your first draft is, the more likely someone will get annoyed and take it over; in either case, this approach will almost always end with a coherent business strategy getting written.

The simplest business M&A strategy is building everything in-house and doing no acquisitions. Generally, I believe that should be the default strategy for most small companies, but the possibilities expand as your business grows. For example, common M&A strategies include:

- Acquiring revenue or users for your core business (for example, Match Group's acquisition of Hinge).
- Entering new business lines via acquisition (for example, Stripe acquired Index to power point-of-sale devices).
- Driving innovation by acquiring startups in similar spaces (for example, Facebook's acquisition of Instagram).

- Reducing competition. Finally, you will absolutely never see this in writing, because no one wants to show up in court documents, but many acquisitions are motivated by reducing future competition. An example of this is Google's 2013 acquisition of Waze, which prevented potential future competition on mapping. If you simply cannot figure out what the real M&A strategy is, ask yourself whether it's about reducing competition.

Essentially, any strategy is reasonable as long as it gets written down, and the executive team is aligned around it. Conversely, if it's not written down or the executive team isn't aligned, it's very difficult to move forward to the next step: applying that business strategy to a particular acquisition.

ACQUISITION THESIS

Your acquisition thesis is how a particular acquisition fits into your company's business strategy. In particular, your acquisition thesis should answer one complex question: what, specifically, needs to be true for this acquisition to fit into your business strategy?

Your answer needs to cover product capabilities; intellectual property; their book of business; revenue; cash flow; whether it's a business acquisition (purchasing the entire business for revenue and product), product acquisition (purchasing for product, generally to increase your existing revenue through your existing distribution), or team acquihire (purchasing for the team members themselves); and every other aspect. The goal is to identify the assumptions that need to be true for the acquisition to be worthwhile.

This thesis will greatly shape how you value a given acquisition. Consider how differently you'd evaluate these two scenarios for a potential acquisition:

- You run a code hosting business. A competitor is capturing market share with their integrated CI/CD offering. You're building a competing CI/CD but are 18 months away from delivering it. You decide to evaluate acquiring a startup with CI/CD offerings. The product works, but you're skeptical of their ability to scale long term.
- Same scenario, but you haven't started building an offering, and think it would take 36 months to get to market.

In the first scenario, the advantage is exclusively in time-to-market. You'd be evaluating reducing revenue impact from the competitive threat until you delivered your internal offering. You don't even need to worry too much about

the product's quality: it doesn't need to scale forever, just for 18 months. In the second scenario, the thesis' validity depends on a product that can scale without replacement. This means you'll want to dig much deeper into the implementation, infrastructure, and underlying costs.

Documenting your acquisition thesis is extremely valuable. It serves as an input to the next stage—the Engineering evaluation—and prevents significant misalignment among the deal team. The messy incentives often lead to scenarios in which part of the team views the acquisition as an interim solution, and another part values it as a permanent business or product pillar. It's exceptionally hard to come to the same conclusion when reasoning about value from different perspectives.

As you articulate the acquisition's value, you also have to think about valuation. One challenge here is identifying an appropriate valuation without getting caught up in the valuations driven by venture capital during booms. Venture's risk model allows it to miss nine times as long as it gets the tenth one right, but as an acquiring company you can't simply write down an acquisition; you're also on the hook for the costly integration necessary for it to have a chance to succeed.

ENGINEERING EVALUATION

Once you have an acquisition thesis that identifies the necessary assumptions to justify your acquisition, the next step is for the deal team to split off to validate those assumptions. Your Finance team will focus on revenue forecasts, your Legal team will focus on intellectual property, and within Engineering you'll need to focus on validating assumptions around implementation complexity, integration, scalability, security, compliance, Engineering costs, and Engineering culture.

The general approach here is as follows:

1. Create a default template of topics and questions to cover in every acquisition.
2. For each acquisition, fork that template and add specific questions necessary to validate the specific acquisition's thesis.
3. For each question or topic, ask the Engineering contact for supporting materials that you can review before meeting, such as their SOC 2 certification, API documentation, current team structure, and so on. They likely won't give you everything you ask for, but whatever they do provide will allow you to narrow your focus.

4. After reviewing those materials, schedule a one- to two-hour Engineering discussion with the Engineering contact, with the aim of either addressing or scheduling a follow-up for all yet-to-be-validated assumptions. You will have limited opportunities to meet with the other team, so cram as much as possible into each of these meetings. Don't assume you'll have six meetings; plan for having two.
5. Run the follow-up actions, validating or invalidating assumptions along the way.
6. Sync with the deal team on whether it makes sense to move forward.
7. Potentially interview a few select members of the company to be acquired. This is very common if it's an acquihire, but relatively less common if it's a business acquisition. I mostly recommend this for individuals who you expect to fill a senior role in the combined organization. The purpose of the discussion is to protect them as much as it is to protect you as the acquirer: you don't want to put yourself or the new leader in the position where you tell their new direct reports that you never interviewed them! Outside of an acquihire, I don't recommend interviewing most of the folks; it's time-consuming and a worse signal than reviewing their existing work.

There are a few more steps in the Engineering evaluation process, but they generally skew toward the superficial. For example, most acquisition processes will include a product demo, but for whatever reason these demos tend to be painfully generic—you can usually learn more by using the product for a few minutes (which I highly recommend doing). Certainly, go along with the default steps to be polite, but don't expect them to be very informative.

Throughout this process, you'll have to decide how many people to involve from within the Engineering team. This depends on the size of your team, the size of the company being evaluated, and so on. Generally, you want to involve as few people as possible. Acquisition discussions are, inevitably, very distracting and rarely end in an actual acquisition, so involving more folks is usually painful without much of a return. However, sometimes there are areas where you simply don't have enough detail to evaluate the company effectively, and that's where it pays to pull a few more people in (especially when it comes to compliance and security questions).

Recommended Topics for an Engineering Evaluation Template

Many of the questions you want to answer about a given acquisition target will tie into your acquisition thesis, but often the acquisition thesis will miss obvious sources of risk. Sure, JPMorgan made sure to model out the potential value of Frank's acquired data, but they didn't verify whether that data came from a real production database. To avoid missing the obvious, you should create a base evaluation template, and run each particular evaluation from a forked copy of that template with acquisition thesis-specific topics added on top.

Topics I would recommend including in your base evaluation template are:

Product implementation

Particularly, you want to verify that the implementation is real and that it is a first-party implementation rather than powered by a third-party capability. For example, *OpenAI* has spawned a flurry of AI-driven startups whose implementation is primarily calling OpenAI's APIs, rather than intellectual property held by the startups themselves. It's particularly valuable to have an engineer, potentially you, read the actual source code of the to-be-acquired company. Companies will be very reluctant to share their actual source code but will be open to an engineer showing the code over a video call. Even reluctant companies will rarely block an acquisition from moving forward solely by refusing to share some amount of their code.

Intellectual property

Similar to the above point on product implementation, while your Legal team will generally dig into any patent portfolio, on the Engineering side you should dig in to uncover whether the intellectual property valued in the acquisition actually exists. Once again, reading the actual source code is particularly valuable.

Security

Who's responsible for security at the company, and what are they doing? Are they doing routine penetration tests, and where are their latest results? Are they doing periodic security training for their employees, and what do they cover? Do they know of any

breaches? If they were breached, how would they know? It can be difficult to get answers to these questions in many early-stage startups, but any sizable company should have a team that can answer these questions directly. If they don't, and they have highly valuable data (health data, financial data, and so on), then you should keep in mind that they may have already been breached and simply don't know it.

Compliance

What compliance certifications does the company have? SOC 2 Type 2? HITRUST? Nothing? To the extent that they claim a given certification, have they kept up with the audits? What did their last audit report show? It's common for companies to say they have some certification, say SOC 2 Type 2, but to have no audited reports to share from the past two to three years is something that merits investigation. Particularly within a small deal team, you might be the only person with context on compliance, and it can cause a great deal of pain if not caught until after completion (or, at minimum, it can be a factor in changing the relative valuations of the two companies if caught early).

Integration mismatches

Who are the vendors, particularly their primary cloud vendor (AWS, GCP, Azure, on-prem, and so on) and core technology stack (programming languages, storage tiers, and any vendors used in their production stack)? You are looking for the extent of the overlap, such that you can answer whether integrating their stack with your existing stack will be relatively painful (best case) or nearly impossible (worst case). Particularly, look for multi-year contracts that might inhibit rapid consolidation, or at least increase the costs of doing so.

Costs and scalability

To the extent that you plan to continue running the current stack, how scalable and operable is it? How much does it cost to run relative to each dollar it generates? It's possible to separate costs and scalability, but they tend to be most interesting when evaluated together. Ideally, you want to see a spreadsheet of all Engineering costs over time, along with revenue and a key user engagement

metric. This will make it clear whether there are economies or dysfunctions of scale, which will help you understand the product's long-term revenue and cash-flow contributions. If this spreadsheet doesn't already exist, you should be able to sit down with the Finance team working on the deal to estimate what these numbers look like.

Engineering culture
: The acquired company's engineers will become your teammates, and you'll want to understand how it will feel to work together. How does the company make technology decisions? How do they structure teams? Does the Engineering contact mention other people, or primarily speak of themselves as the decision maker? There's no right or wrong here, but you do want to understand how their operating model differs from yours, and whether you're willing to deal with that friction.

Whatever list of topics you start with, keep extending it after each acquisition evaluation. Over time, this will become a source of significant value to your company. Keep in mind, you can indeed go a bit too far in these investigations: at one point, I was silently dropped from Stripe's acquisition process for asking a few too many questions.

Making an Integration Plan

A surprising number of acquisitions are completed with only an informal integration plan, which leaves a lot of room for miscommunication between the acquiring and acquired companies. My experience is that it's well worth your time to develop a clear, written plan. Here is the baseline integration plan that I'd recommend starting with, which acknowledges the many known and unknown factors involved in integrating a real business with another real business:

- Commit to running the acquired stack "as is" for the first six months. Let the acquired company know that you will aim to consolidate technologies wherever feasible.
- Bring the acquired Engineering team over, with their head of Engineering reporting into your head of Engineering (or the appropriate business unit's engineering lead if they are a small acquisition contained within a single

business unit). Let the acquired Engineering leader know that you will aim to combine vertical teams (developer productivity, infrastructure, and so on) as feasible.

- Be direct and transparent with any senior leaders about roles where they could step in, and how you would make that decision to maximize their success along with the existing team's. Even if someone is deeply qualified, you'll better set them up for success by letting folks get to know them a bit before inserting them as a new boss.

I prefer the preceding integration plan because any other details you decide upon will almost certainly be specifically wrong, and you will have to remake them along the way. However, even if you use a loose plan like the one outlined here, you'll learn a tremendous amount about the opportunity and opportunity cost by planning a few steps deeper into how the acquired company would integrate into yours, and I highly recommend pushing on the ideas until you identify the biggest risks.

As you work through the evaluation process, you'll also need to develop a point of view on a number of integration choices—specifically, identifying the most successful approach to integrating the acquired team, product, or business into your existing business and function. The three most important questions to work through are:

1. How will you integrate the technology?
2. How will you integrate the teams?
3. How will you integrate the leadership?

Once you've figured out these things, document them and validate them with your deal team. Until you socialize your plan, it's likely that entirely different plans have been communicated. Sharing your perspective on these integration choices will allow you to pull folks onto the best plan to move forward post-acquisition.

TECHNOLOGY INTEGRATION DECISIONS

The details of technical integration will depend on the value you hope to create. The two extremes are a full rewrite on one end and running in full separation on the other. Many of Google's smaller acquisitions led to full rewrites, such as the Metaweb acquisition in 2010. Metaweb had to spend years reimplementing their working technology into the Google stack. On the other hand, Google's largest

acquisitions, like Waze or YouTube, ran (at least initially) on their own stacks and facilitated integrations through APIs.

Your technical integration should be guided by your Engineering strategy. In the strategies that I've operated in, we knew we would always eventually converge on a single cloud environment (e.g., AWS versus Google Cloud), and generally tried to converge tech stacks as early as possible. Even in cases where it was technically feasible to operate separate stacks, you'll end up with two distinct Engineering subcultures with different modes of technical decision making. Most companies won't tolerate multiple subcultures until the company grows quite large (e.g., 1,000 or more engineers) and teams work in very different problem spaces. After that point, they usually are initially tolerant of subcultures. For example, Uber's Autonomous Technologies Group ran separate stacks successfully for a period of time, given that they focused on significantly different problems than the company's core product.

TEAM INTEGRATION DECISIONS

The second integration question to dig into is about structuring teams. This will depend heavily on the relative size of the two companies, but generally you'll either have the acquired company's head of Engineering reporting to you or to the Engineering lead for the business unit they're joining. You may also wish to separate infrastructure, data, or security teams to report into your existing, centralized teams. None of these approaches are obviously right; the best approach will depend heavily on the details and experience of the leaders involved.

LEADERSHIP INTEGRATION DECISIONS

Deciding what to do with the acquired company's leadership is always tricky. Often, you'll find that the Engineering leader at an acquired company has been struggling in their role, or struggles to imagine themselves reporting to someone else running Engineering. The opposite happens, too, with excellent leaders who are energized by partnering well with you, who are glad to teach you, and who would have been difficult for you to hire directly. Most importantly, think about balancing doing the right thing for the acquired company with doing the right thing for your existing team. Success depends on both groups working together effectively, and it's easy to dig a deep hole if your existing team feels the acquired team is underqualified for their new roles.

Dissent Now or Forever Hold Your Peace

As discussed earlier, the incentives around any acquisition are very messy. When you consider only what could go right, acquisitions are irresistible. Some of the industry's defining products blossomed through acquisition, like Meta's acquisition of Instagram and WhatsApp, or Google's acquisition of Waze. There's also Match Group's dating rollup strategy, acquiring Tinder, Hinge, and many others. Acquisitions can be transformative.

However, even ignoring opportunity cost, most acquisitions don't deliver on their intended impact. Think of Google's acquisition of Motorola, eBay's of Skype, or Amazon's of Goodreads. Further, the Engineering team will absorb most of those opportunity costs, and it's your responsibility as an Engineering executive to inject realism into the process.

Sometimes this is pretty difficult, and won't make you friends. My best advice is to anchor your feedback to the company's goals, rather than Engineering's, and to make sure your concerns are truly from the company perspective. Ignoring the philosophical question of whether it's fair, Engineering will incur much of any acquisition's operating cost and complexity and owes the company (and itself) a strong, clear point of view.

Engineering will often have one vote out of many, rather than an outright veto over the acquisition, and the wider group won't always agree with you. Sometimes, being an executive means doing something painful for your function that may be impactful for the broader business. I will say that I've rarely regretted being overruled on these things. Usually, I was missing an important consideration. These things have a surprising way of working out for the best, even when it seems like they shouldn't.

Being Acquired

I want to end with a few words on the inverse challenge: getting acquired by another company. There's a lot to be said about how to get acquired, which I think Touraj Parang's *Exit Path* (McGraw-Hill) covers nicely, so I'll focus on navigating the process from an Engineering perspective.

The key thing to understand is how the acquirer views the acquisition. Do they view it as a business acquisition, product acquisition, or an acquihire? If it's a business acquisition, then changes are likely to happen slowly. If you talk to folks at Slack post-Salesforce acquisition, or LinkedIn post-Microsoft acquisition, you'll find that changes have definitely happened. But, for most people, their lives didn't change much overnight.

In the other two cases, things are likely to change very quickly. An acquihire is essentially starting a brand new job where you already know a few folks. A product acquisition will start with an integration to prove out the acquisition thesis, often followed by reworking the implementation to follow the acquiring company's standard development process.

The best thing you can do for the team is fight for their compensation packages before the acquisition finalizes, and then push them to acclimate to the new world. Fighting the new world will only cause chaos—it won't accomplish anything valuable. There's no question that the changes will happen either way, but each staff member has an opportunity to be perceived either as a collaborative member of the team or someone anchored to the past.

Finally, as an executive, you have to decide for yourself if it makes sense for you personally to move forward with the acquisition. Even if you have an equity acceleration trigger in your contract, you may be obligated to join the acquiring company for the acquisition to complete, a role that is sometimes referred to as a "key person" in acquisition discussions. If that happens, I'd encourage you to look for the opportunity for yourself. Acquisitions often accelerate careers and open up unique roles that wouldn't be available to you otherwise. For every horror story about a bad acquisition, you'll find another, quieter story about a transformed career.

Summary

In this chapter, I laid out an approach to evaluating potential mergers and acquisitions by defining an overarching business strategy, acquisition-specific thesis, and an Engineering evaluation to validate that thesis. This structure, summarized here, will provide a systematic, principled path through the inherent messiness of these discussions:

- Maintain a business strategy that describes how acquisitions should contribute to your company's business.
- Craft an acquisition thesis that articulates how this acquisition fits into your business strategy.
- Document the Engineering evaluation of each acquisition to ensure your executive team has a clear view of both the opportunity and the risks.

I also shared my advice for executives on the other side of the table: those being acquired. In that scenario, I recommend the following approach:

- Understand how the acquirer views the acquisition.
- Make sure your team gets taken care of, not just the executives.
- Figure out whether there's a compelling role for you, or if your goal is to depart when the acquisition closes.
- If you find yourself obligated to stay through an acquisition but aren't particularly interested in the new role, make sure you're compensated for it.

In both cases, remember that these are complex decisions with messy incentives. Even the best acquisitions will get a bit weird.

Find links to further reading and resources on *https://lethain.com/eeprimer-refs-7*.

Developing Leadership Styles

For a long time, I found the micromanager CEO archetype very frustrating to work with. They would often pop out of nowhere, jab holes in the work I had done without understanding the trade-offs, and then disappear when I wanted to explain my decisions. In those moments, I wished they would trust me based on my track record of doing good work. If they didn't trust my track record, could they at least take the time to talk through the situation so I could explain my decisions?!

I longed for a more distant CEO, one who defined a clear process for how we worked, benevolently approved my headcount requests, and occasionally sent me a note confirming my inherent genius, but otherwise left me to do my work. As I spoke with industry peers, I was surprised to realize that the CEO-at-a-remove does exist. In fact, there are many CEOs who are removed from daily execution, but my peers working with those CEOs weren't celebrating them. Instead, they were quite frustrated. Where I'd imagined an absentee CEO would feel empowering, executive teams working with such CEOs in practice often found they couldn't move important decisions forward. When the executives did make progress, it was by accepting whatever outcome they could build consensus around, rather than making the best possible decisions.

From that realization, I came to more fully appreciate that there's no one style of executive leadership that's applicable everywhere, and that particularly effective executives have a number of distinct styles that they're able to swap among to solve a given challenge. Many line management roles rely on process-oriented leadership. Middle management roles often select for consensus building. And architect roles may depend on definitive, top-down decision-making. As an executive, you need to have the ability to swap across all three styles.

This chapter covers:

- Why executive roles are particularly dependent on having multiple leadership styles
- How and when to use three primary styles: leading with policy, leading from consensus, and leading with conviction
- How lessons taught early in management careers about micromanagement discourage too many executives from leading with conviction
- How to develop leadership styles that you currently feel uncomfortable using
- How to balance across these styles, especially when you're uncertain which is most appropriate for a given scenario

By the end, you'll have a grasp on three distinct leadership styles, know when to use them, and have a path for developing any of those styles that you haven't spent much time using thus far in your career.

Why Executives Need Several Leadership Styles

My favorite chapter in *An Elegant Puzzle* is titled "Work the policy, not the exceptions". As an executive, I still agree with what I wrote there, but I've also come to recognize that every policy has an implicit clause at the end along the lines of, "Exceptions are approved by the CTO."

Good policy empowers your organization to move quickly without you in the room, and is the foundation of an effective organization with 50 or more engineers. However, even with great policy guiding day-to-day operations, executives will find themselves spending a great deal of time handling exceptions. For example, how you *generally* think about Engineering prioritizing business partnerships is amenable to policy, but the extent that you prioritize a given partnership *right now* isn't something policy can answer if your head of partnerships is escalating to your CEO to overturn the current policy.

Further, good policy requires iteration and exceptions often need to be made immediately. Slowing down the business isn't always an option, which then forces you to choose between relying on a quickly created bad policy or treating the situation as a one-off solution. That set of options would have broken my earlier career heart, but it highlights something it took me a while to understand: an effective executive must develop leadership styles to operate with process and to operate within exceptions.

The next sections will walk through three distinct approaches to leadership:

Leading with policy
: For recurring decisions that need to be made consistently by many individuals within your organization

Leading from consensus
: For infrequent decisions with context spread across a number of different stakeholders

Leading with conviction
: For decisions without any clear proposal, where involved individuals are deeply at odds with one another, or that have outsized, long-term impact on your organization

To be a dynamic executive, you must develop all three styles, the ability to apply the right one to a given circumstance, and the self-awareness to recognize when you're leaning too heavily on the style you find most comfortable.

Leading with Policy

Leading with policy is creating a documented, predictable means for making a decision. It's particularly useful for addressing recurring decisions that need to be consistently made by many individuals within your organization, such as assigning performance scores or determining promotions.

Processes typically include a heavy dose of policy ("each manager will use our career framework to assign performance scores for their team during our performance cycle next week"), but you can use policy outside of process as well ("whenever you're considering bringing on a new vendor, these are the criteria you should use to evaluate that decision").

EXAMPLES

A few cases of leading with policy in my career are as follows:

At Calm
: Teams felt that they couldn't get any testing or technical improvement work onto the roadmap and it was causing both frustration within the team and ongoing quality issues. We reworked our planning process to allow Engineering managers to own 20% of their team's roadmap each quarter. This allowed each Engineering manager to be accountable for prioritizing

this work rather than escalating when they disagreed with their team's roadmap.

At Stripe

We came to believe we were relying too heavily on external hires and it was causing cultural drift within Engineering. One of the challenges was the difficulty in comparing internal and external candidates effectively. To address this, we built out an explicit process that included providing feedback for internal candidates who weren't selected. This created a predictable, ongoing policy for handling these scenarios.

At Carta

Promotions felt inconsistent across teams and managers, so we built out a promotion committee to evaluate all Engineering promotions. This did not remove all bias—nothing can—but it meant that all promotions in a given performance cycle were reviewed by the same group of individuals who were highly calibrated on our leveling rubrics.

MECHANICS

The core mechanics of leading with policy are:

1. Identify a decision that needs to be made frequently, and where it's important that a number of individuals make consistent decisions.
2. Study how those decisions are being made today, by individuals who are making them well (this last caveat is important! You want to build around the best existing decision makers, not immortalize the mean decision-making approach).
3. Document that methodology into a written policy, with feedback from the best decision makers whose judgment you want to build into the policy.
4. Roll out the policy. At a large company, validate it with a small group and expand from there. In a small company, you'll likely deploy the policy for everyone.
5. Commit to revisiting the policy with data after a reasonable period (e.g., one performance cycle, five architecture reviews, or whatever makes sense for the particular policy).

Leading from Consensus

Leading from consensus is pulling together the relevant set of stakeholders to identify a shared approach for a given problem. It's most effective for decisions with multiple stakeholders, each of whom has a critical subset of the necessary context to make an effective decision, such as deciding to acquire a company.

Consensus has a reputation for being slow, but I think that's an overstated concern. In situations where no one has the entire relevant context, there's no particularly quick solution to making a robust decision. You can't rely on policy for these decisions because you won't have enough repeat exposure to base your policy on. You can rely on one individual building conviction, but that's generally *even slower* than building consensus. That's because the relevant context tends to be so broadly distributed that it's easier to aggregate via a decision-making group, unless the decision is obviously sufficiently critical that an executive should prioritize their attention on it.

EXAMPLES

Some examples from my experience of leading from consensus include:

After Stripe acquired Index (a startup that developed a point-of-sales system)
: We had to decide how to integrate their technology into our systems. Although there were many individuals with strong opinions, no one individual had the full set of details in their head. We wanted to build a positive relationship with the recently acquired team, so we relied on a consensus-heavy decision-making model to determine the path forward.

At Uber
: There was significant disagreement about selecting a service orchestration solution. Should we use Mesos, Kubernetes (which was, at the time, very early in its development), or build something ourselves? There was no individual who understood all three options well, and all three would require significant customization before they met our full requirements. We worked as a group to build consensus on the path forward.

At Calm
: There was ongoing friction regarding which pieces of content got the best in-app visibility. This was particularly meaningful to the Content team, whose performance was evaluated in part based on their releases' visibility. Conversely, Product and Engineering felt that dynamic was leading us to serve lower-performing content (e.g., increasing performance for a specific

piece of content, but reducing overall performance of our content as a body of work). There was no clear decision maker who could cut these ties, so we relied on consensus to work through the degree of customization we'd support and how those decisions were made.

MECHANICS

The mechanics of leading from consensus are:

1. Any decision without a relevant policy is a candidate for consensus-based decision making, but where consensus shines is when decisions must be made with many stakeholders, none of whom has the full and relevant context.
2. When considering a given decision, evaluate whether it's important to make a good decision. I often ask myself, "Will I remember what we did here in six months?" as a proxy for this. In cases where the answer is "no," make a quick decision and move on. If it's a "yes," then it's a good opportunity for consensus.
3. Identify the full set of stakeholders to include in the decision, pulling them into a shared communication channel (often a shared chat channel).
4. Write a framing document capturing the perspectives you know and encourage other stakeholders to add their perspectives. For urgent decisions, this can be very brief, but I've found that a bit of documentation is usually worthwhile.
5. Identify the most senior, engaged party (or parties) to decide how the group will work together on the decision. Particularly important is aligning on the deadline for making the decision.
6. Follow that leader's (or leaders') direction on how to build consensus in that situation.

Note that, as an executive, often the most senior, engaged party will be you. In that case, you get to design the consensus-building process to ensure key stakeholders feel involved, or task an empowered delegate to handle it for you (e.g., a senior-most engineer, one of your direct reports). My advice is to emphasize two ideas. First, make sure that all stakeholders feel heard. Second, set a tight timeline and hold to it. These two ideas balance building buy-in and moving quickly, which is the key trade-off in building consensus.

Leading with Conviction

Leading with conviction is pulling all the relevant context into your head, thinking through the trade-offs, and personally making a definitive decision. It's most useful in scenarios where no one has a clear perspective on making a decision, where the involved stakeholders have wholly incompatible views on the path forward or on how to make decisions with significant, long-lasting consequences (e.g., spinning up or shutting down a business unit).

This is an expensive leadership style because it requires significant executive involvement, and I find that many executives either use this approach too frequently (leading with conviction on almost all decisions) or don't use it at all (view it as too deep in the details for someone of their seniority). Both are bad signs. Particularly during periods of significant change (e.g., downturns, or times when you land new funding or are expanding into new business lines), this is a leadership style you should be leaning on frequently.

EXAMPLES

A few times when I've focused on leading with conviction were:

Soon after I joined Calm

I began to suspect that our migration from monolith to services had failed without acknowledgment. I dug in to understand the differing perspectives and made the top-down decision to reverse the migration and return to our monolith. Several smart engineers disagreed vehemently, but I knew there was no way to make progress on such a contentious decision other than building personal conviction and owning the decision myself.

At Stripe

I needed to fill a key leadership position and there was significant tension between hiring external and internal candidates, as well as over the final internal candidate. As mentioned earlier, I designed a clear process to evaluate and gather feedback on the candidates, but ultimately there was still significant disagreement on determining the final candidate for one particular role. I built conviction in the candidate to move forward with and moved forward with them.

At Uber

The Infrastructure team was getting overloaded by requests to provision new services, and we were falling behind a bit more each week. Teams were stressed but lacked any clear path forward. I dug into the issue and decided to build a self-service provisioning tool that scheduled into an overutilized, shared cluster of servers in our data center. This wasn't perfect, but it allowed us to get out of the way for the long tail of new services that never got meaningful usage, rather than being an upfront gatekeeper. It also allowed the team to unblock rather than wait for long-term direction.

MECHANICS

The mechanics of leading with conviction are:

1. Identify a meaningfully important scenario in which no one has the full context to make a high-quality decision, and where the decision is unlikely to repeat (e.g., performance review happens a lot, a specific acquisition happens exactly once).
2. Figure out the individuals with the most context on the topic and go deep with them to build out your own mental model of the problem space.
3. Pull that context into a decision that you write down.
4. Test that decision widely with folks who have relevant context (in many cases, this is a good opportunity to lean on members of your external network with relevant experience, although probably not in the case of particularly sensitive decisions such as acquisitions).
5. Tentatively make the decision, communicating to stakeholders that you've made a tentative decision that will go into effect a few days in the future (try to keep the timeout to a week at most to avoid having to litigate the decision multiple times).
6. Finalize the decision after your timeout and move forward to execution.
7. One downside of leading with conviction is that it can be hard for others to understand your decision. Avoid that outcome by writing down your decision-making process!
8. Finally, take a deep breath, acknowledge that even the smartest decisions can end up being mistakes, and move forward.

ISN'T THIS MICROMANAGEMENT?

One of the very first lessons we teach new managers is to be wary of micromanagement. This lesson is so strong for some leaders, that it remains an important leadership principle for them even as they become an executive. When I push such executives to proactively inspect their team's work, they'll often earnestly reply that they've actively decided not to inspect their team's work because they don't want to be a micromanager.

I sympathize with that anxiety, having experienced it in my own leadership journey. My early years as a manager were best characterized as having a number of micromanagement tendencies that I made myself slowly unlearn. One way I worked against my instincts was to be less aware of some day-to-day particulars. My rationale was that if I didn't know what was happening, I was less likely to get caught up trying to optimize it. I justified this approach in "Introduction to Systems Thinking", and internalized Michael Gerber's core message in *The E-Myth Revisited* (Harper Business). My rationale was that leaders work *on* the system, not *in* the system. This distanced approach became very comfortable at the time, but it ultimately undermined the teams I had intended to support.

When those teams needed help, I didn't have much context to help them, and I certainly couldn't detect them running into trouble before they could. If they ran into conflict with another team, I could *only* rely on consensus and policy, as it's impossible to build conviction without engaging in the details of the work at hand. I came to appreciate that being too far removed was just as disempowering as being overly engaged.

It's not micromanagement to know what a team is doing, nor to question the thinking behind a decision they're making. It's reasonable to review their goals, and their progress against those goals. It's OK to talk to their internal and external customers and hear feedback on how it's going. Those are all activities that empower your team to do meaningful work. If you ever have someone imply that those activities are micromanagement, then it's far more likely that the individual is misaligned with you than that you are too deep in the details. Don't let accusations of micromanagement steer you away from doing your job.

To evaluate whether you *are* micromanaging, ask yourself two questions:

1. Are you carefully considering the feedback of those closest to the work before making decisions?
2. Are you adding friction to a team that would have gotten to a similar quality decision without your involvement?

As long as you can honestly answer "yes" to the first and "no" to the second, then don't spend more time worrying about it. If someone's accusing you of micromanagement, you do have a problem, but it's not a problem of being too far into the details.

Development

Most executives lean heavily on either policy or conviction, and view consensus with suspicion. Similarly, executives who started their journey as founders or have mostly worked in small companies may view consensus-building as inefficient, and they may be convinced that it's not a useful leadership style to develop. Executives who developed their styles in large organizations may lean on consensus too frequently, even when they have enough context to take action directly.

Whichever of those buckets you fall into, I recommend these steps for getting more comfortable with leadership styles that you use less frequently:

1. Roughly once a month, set aside an hour to collect the upcoming problems you need to resolve.
2. Within that list, identify a problem that might be solvable using a style that you don't use frequently or are less comfortable with.
3. Do a thought exercise of solving that scenario using that leadership style.
4. Review your thought exercise with someone—either inside or outside your company is fine—who you've seen use the style effectively in the past.
5. Then think about how you'd address the same scenario using a style you're more comfortable with: what works better or worse?
6. If the alternative isn't much worse, and the stakes aren't exceptionally high, then go ahead and use the style you're less comfortable with to resolve the scenario.

The mechanics of improvement are straightforward. It's really that simple; the hard part is setting aside time to do it and following through. If that's where you're getting caught, find someone else who is trying to also expand their repertoire of leadership styles and hold each other accountable for practice.

Balancing Leadership Styles

You've gotten some general guidance for when to use each of these leadership styles, but how do you balance them? You should lead with policy to streamline recurring decisions, lead with conviction for non-reversible decisions with significant uncertainty, and so on. In most cases those guidelines should be sufficient to select between the various styles, but I find that most executives have a strong tendency to rely on a preferred style, and it's important to recognize situations in which your own default may lead you astray.

Most importantly, you should be particularly wary of maintaining your default as your context changes. When I first joined Calm, I was still ramping up on the business and relied heavily on leading with policy to cover for my gaps in context. I kept relying on that a bit longer than ideal and came to appreciate there were places I needed to lead with conviction to unblock execution. I would have been even worse off if I'd continued defaulting to policy after joining Carta, as the organization had already done an excellent job of using policy effectively, and I needed to contribute by merging in another leadership style rather than providing more of what they were excellent at.

Many executives keep relying on the same default style as their companies go from finding product market fit to rapid expansion, or as those companies go from rapid hiring to relearning to operating post-layoffs. If something significant has changed around you, but you're still defaulting to the same style, it's worth spending some time thinking about why.

Finally, even if nothing has changed, if you're letting some styles atrophy, think about finding a few places to branch out to continue developing yourself. If you get too rusty at any of them, you'll have to pick them back up under duress, which is surprisingly difficult because it's not just *you* who has to get comfortable with the different leadership style; everyone who works with you will need to learn to adjust as well.

Some Reasons Executives Get Disengaged

Sometimes executives rely on consensus or policy because those are their preferred management styles, but I've also found that executives may end up in those styles because they take less context to operate and the executive has become disengaged from their work.

I've generally found that executives become disengaged for three broad reasons:

1. External demands have broken the executive's work systems. This might be an aging parent, a young child, or a personal sickness. Their previous work habits are no longer supporting their success, and they haven't built new habits to compensate.
2. The executive's pursuit of "enlightened distance" to avoid micromanagement has gone too far. Instead of digging into the details, they tell their team that they trust them and encourage them to follow what makes sense to them. This culminates in the manager ignoring their team rather than empowering it.
3. The executive is chasing energy and will continue to drift toward wherever energy accumulates in their life. They might fixate on short-term fires and lose sight of their team's core work. Or they might become very active or loud in their angel investing because they've become bored with their work.

As you think about identifying these signs in yourself, the first two categories—spike in external demands and enlightened distance—are the sorts of things that can be diagnosed and solved by a helpful manager. Admittedly, as you get further into your career, you're increasingly unlikely to have a helpful, attentive manager, but it's still likely that a peer will nudge you ("it feels like you've been a little distracted recently") or that you can self-diagnose ("how would I know if my team was really struggling right now, without someone proactively telling me?").

The final category, disengaged because you're chasing energy elsewhere, is the hardest to resolve, and often requires some messy trade-offs.

Mihaly Csikszentmihalyi has written two delightful books: *Creativity* (Harper Perennial) and *Flow* (Harper Perennial). The latter is particularly helpful in introducing a chart that shows the relationship between a participant's skill, a scenario's challenge, and the participant's likelihood of reaching the flow state (see Figure 8-1). An expert programmer, for example, is unlikely to attain flow when doing rote programming work. A beginning programmer is unlikely to attain flow when doing extremely difficult programming work. Flow is most easily achieved when experiencing a challenge that is at, or just slightly above, your current skill level.

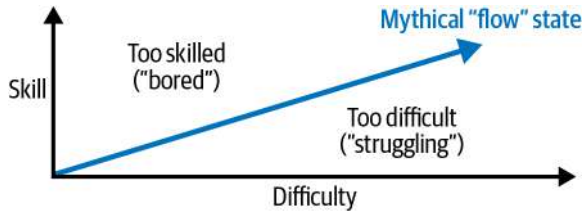


Figure 8-1. Intersection of skill and difficulty is "flow" state

So, too, it goes for executive engagement. If you start looking elsewhere for energy—whether you're falling behind on core work duties or getting a bit too focused on angel investing—it's almost always because your role is too challenging, or not challenging enough.

If you find yourself undermatched by your role, try to supplement with a new kind of work. For example, identify an interim assignment supporting an additional team, or ask the CEO to assign you to a major business priority. If you're overmatched by your role, the playbook is the same as it would be for someone you're managing within your team: identify and address your gaps. As an executive, your manager is generally the CEO who's usually too busy to directly assist, but they can fund a coach, and you can always ask your peers for help.

Summary

In this chapter, we talked through three distinct leadership styles—leading with policy, from consensus, and with conviction—and how to use them. We also covered how to determine when to use each of these styles, and how to develop any that you aren't yet comfortable using.

In my experience, almost all executives have a principled perspective on why one of these leadership styles is undesirable. We discussed why leading from conviction is often framed as micromanagement, but also how working for CEOs without conviction leads to slow, muddled execution. Many executives will make a similar argument against leading with policy, as you can indeed lead poorly from policy.

The key insight to remember is that you can use any leadership style poorly, but that doesn't mean it isn't useful when done well. Don't let the worst cases of any given style convince you that the style itself is useless. Instead, look for better examples of those styles to learn from, find ways to practice the uncomfortable parts, and continue growing yourself into a dynamic leader.

Find links to further reading and resources on *https://lethain.com/eeprimer-refs-8*.

Managing Your Priorities and Energy

Back when I was managing at Uber, I latched onto a thinking tool that I drilled into the teams I worked with: reach the right outcomes by prioritizing the company first, your team second, and yourself third. This "company, team, self" framework proved to be a helpful decision-making tool, and at the time I felt it almost always led to the correct decision. It also helped me articulate why I disagreed with some of my peers' decisions that violated this hierarchy by placing individual or team preferences over the company's priorities.

As I've become a more experienced manager, I've stopped giving this advice. I still believe it's conceptually good advice and I continue to see managers who fail because they are missing this perspective. However, I've also seen some of the best leaders that I've worked with burn out by following this advice too loyally. A long-term career depends equally on being impactful and staying engaged.

In this chapter, we'll discuss:

- How I previously used the "company, team, self" framework to prioritize
- How energy management is a positive-sum process, and why that's changed my model for prioritization
- How I've moved to an "eventual quid pro quo" framework
- Distinguishing between self-interested behavior and the appearance of self-interest
- Why it's important to remain flexible, even when using the best framework

Let's drill into the details.

"Company, Team, Self" Framework

As I mentioned, I used to rely on the "company, team, self" framework to check my decision making, particularly during my time at Uber. Uber Engineering had a strong *Not Invented Here* bias, and frequently invested in creating its own technology, such as the M3 metrics system or the Mezzanine sharded storage system. While many of these systems addressed very real scalability concerns, they also tended to get their proponents promoted, and over time it became challenging to distinguish whether any particular platform was principally anchored to business impact or career progression. In that environment, pushing the team to explicitly prioritize the company and team first was extremely valuable.

Some of the concrete scenarios in which I found this framework useful were when:

- Deciding whether to prioritize an interim orchestration solution before a planned Mesos-based orchestration system was rolled out. Yes, we should build an interim automated solution to support Uber's engineers while we waited for the replacement, even though it was viewed as less promotable work than working on the delayed replacement.
- Fixing hiring loops, when stressed-out hiring managers wanted to hire candidates who didn't meet our hiring bar. Yes, you do need to make a hire but, no, making this hire isn't the right outcome for the company.
- Evaluating the relative priority of staffing requests within the organization. Yes, every team would benefit from additional staffing but which would be most valuable to the wider organization?

The framework was particularly helpful within Platform and Infrastructure teams, where our distance from end users and revenue often obscured answers that might have been more obvious to user-facing teams with access to the loud signal of user feedback.

However, the more I followed the "company, team, self" approach," the more I appreciated its biggest downside: the most valuable work at a company is rarely the most interesting nor—oddly enough—what the company itself particularly values. I frequently noticed cases where engineers did the best thing for the company, solving an urgent problem with little staffing, but weren't recognized or promoted for their work. Those engineers would slowly become de-energized and frustrated. As this pattern became clearer, it also became clearer to me that always prioritizing the company's needs was too literal a solution, and a durable

approach would require balancing the company's priorities with the need to remain personally energized for the long haul.

Energy Management Is Positive-Sum

People are complex, and they get energy in complex ways. Some managers get energy from writing some software. That's great, particularly if you avoid writing software with production dependencies. Some managers get energy from coaching others. That's great. Some get energy from doing exploratory work. Others get energy from optimizing existing systems. That's great, too. Some get energy from speaking at conferences. Great. Some get energy from cleaning up internal wikis. You get the idea: that's great. All these things are great, not because managers should or shouldn't program/speak at conferences/clean up wikis, but because folks will accomplish more if you let them do some energizing work, even if some of that work isn't very important.

Rigid adherence to any prioritization model, even one that's conceptually correct like mine that put the company and team first, will often lead to the right list of priorities but a team that's got too little energy to make forward progress. It's not only reasonable to violate perfectly correct priorities to energize yourself and your team, but modestly violating priorities to energize your team in pursuit of a broader goal is an open leadership secret. Leadership is getting to the correct place quickly; it's not necessarily about walking in the straightest line. Gleefully skipping down a haphazard path is often faster than purposefully trudging down the safest path.

There are, of course, rules to breaking the rules. The most important being that your energizing work needs to avoid creating problems for other teams. If your fun project is prototyping a throwaway service in a new programming language, then hmm, maybe that's fine. But if you put it into production, your energizing detour is going to be net negative on energy generation after other teams are pulled in to figure out how to support it.

Correctness and humans mix in complex ways. The most important lesson I've learned as I've become a better manager is that there is almost always a "correct" answer, but applying that answer to your specific situation will always be nuanced and messy. Further, the correct answer is often different depending on whether you're taking a short-term or long-term perspective. Every specific decision is nuanced and complex, but you'll be a better leader if your decision making modestly factors in your team's energy rather than ignoring it.

Eventual Quid Pro Quo

These days, my framework for personal prioritization merges the business-first perspective of "company, team, self" with the belief that becoming de-energized or disengaged is my biggest risk in any particular job. I think of this new framework as "eventual quid pro quo."

The core framework is:

- Generally, prioritize company and team priorities over my own.
- If I'm getting de-energized, artificially prioritize some energizing work. Increase the quantity until equilibrium is restored.
- If the long-term balance between energy and proper priorities can't be achieved for more than a year, stop everything else and work on solving this (e.g., change your role or quit).

Arguably, this is a small tweak to my earlier framework but it's been very important for me, which is the key component of a successful framework. What makes it useful for me is that I am a very literal interpreter of frameworks and this gives me space within the framework to maneuver and adapt. I can remain faithful to the framework without draining out my energy in the first few years in any given role.

Just because it's the right framework for me doesn't mean it'll be the right one for others. To develop an effective prioritization framework requires getting to know yourself (what work energizes you) and your priorities (career, financial, and otherwise). Until you understand those answers, no premade framework is going to produce the correct answers for *you*.

You've probably worked with someone who operated in a short-term quid pro quo mode. This person may have frequently asked what the benefit was to them to take on a given project. That approach does not work well in leadership roles. In particular, it adds friction at the wrong moment, when the requester (typically your boss) is trying to solve the problem. Over time, short-term demands will often mean you're considered last for valuable assignments. Decoupling timing of repayment lets you keep access to interesting work while also pushing to ensure you're rewarded for that work in an appropriate currency. This isn't perfect, as sometimes you won't be rewarded for some work, but it's generally better than the alternative if you are in, or pursuing, an executive role.

Mirrors of Misalignment

One challenge with frameworks is they sometimes don't tell the entire story. As I get further into my career, I'm acutely aware of moments when my actions are perceived as in conflict with some of the earlier mental models I've used to evaluate work. For example, I've recently spent a fair amount of time threading a needle through Engineering to prioritize large language models through our execution. If I were sitting in many of my previous roles, I would judge myself as prioritizing misaligned work. In my current role, I appreciate that there is organizational risk if we entirely ignore potentially-but-not-necessarily transformative new techniques.

I think of this dynamic—when I prioritize work that my previous frameworks would have considered misaligned—as navigating the mirrors of alignment. There is always another layer of context that validates or invalidates any given piece of work. This is why operating from a place of curiosity is so powerful: folks who appear to be doing something nonsensical almost always have more information than you, or are missing a key piece of information that you have.

The place where I see these mirrors most frequently is in corporate planning and projects that bypass the corporate planning process. It's very common for companies in dynamic markets to spend a month planning the nuanced details of their next six months, then immediately invalidate that plan by changing their priorities. This infuriated me for a long time, and my framework thinking labeled the offenders as bad planners. Now I see myself doing the same thing, and I've come to appreciate that many of the folks I silently accused of malpractice were balancing contexts that I had no idea existed.

As an executive, each time this happens is a reminder to share your context more widely and remain curious. You'll rarely be missing strategic context but you'll increasingly be missing the tactical context around how things actually work. Seniority creates just as many gaps in knowledge as it closes.

Orthogonal but Not in Opposition

As you think about expanding your framework to create space for energizing work, the most important guideline to remember is that while it's OK to do a small amount of work that's orthogonal to the company's needs, you will always regret doing work that is opposed to those needs.

Following are some examples:

- Many folks do more public speaking, perhaps at conferences during working hours, than their company strictly values. Doing one or two speaking gigs a year may not directly help your company, but it's also hard to argue that it's causing any negative effects. (Whereas if you did eight or ten speaking events in a year, it would be easy to argue that you're encouraging others in your organization to focus away from the work at hand.)
- If you push to use a new technology on a project mainly out of interest in learning that technology, then you're creating significant execution risk and maintenance costs for your company. This might be energizing for you, but in most cases it's likely in opposition to the company's needs.
- Doing some angel investing in unrelated business verticals won't quickly teach you a tremendous amount of relevant information, but it will slowly broaden your horizons across the industry. Further, it won't take too much time to do. (Doing a tremendous amount of angel investing, or running your own syndicate, certainly might cross a line into being opposed to your core work.)

Almost any outside activity is fine when done in moderation. Almost any decision to use work projects as an opportunity to develop yourself by using non-standard tools is a bad choice. If you're not sure, ask a friend, but it's probably a bad sign that you're uncertain.

Remain Flexible

If you strongly disagree with the idea of factoring in engaging work that is imperfectly aligned with your company's goals, I'd push you to remember that humans aren't robots. That's surely obvious advice but I really struggled with that idea for a long time. It was very frustrating to me that the correct answer to most prioritization and architecture projects was obvious (e.g., why is it controversial that we should use boring technology?), but teams so frequently did something incorrect instead. What folks may not understand is that, for a certain type of

person, strictly adhering to the correct path is very energizing. That kind of person, a person who I used to be most of the time (and still revert to today when particularly frustrated), doesn't need to do sub-optimal energizing work, because doing the correct work is inherently energizing.

However, I admit that I've never really seen high-adherence personalities succeed in senior engineering roles. Executive roles require addressing uncommon circumstances while working with many different people, and only a relatively small subset of them will be energized by adherence. It's particularly rare to find peer executives who are motivated by adherence. In some brilliant moments, you'll get to work with other executives with the same values, and those will be some of your most rewarding moments. But you have to learn to mute those values when they're interfering with forward progress.

When an exceptionally talented engineer or manager is struggling to make progress in their career, their view on adherence is one of my first guesses for why they aren't progressing. It's surprisingly common that they're stuck because they are someone who is energized by adherence, views a lack of adherence in others as a character flaw, and consequently alienates those they need to build relationships with to succeed. Blunting my desire for adherence has been one of the most valuable lessons I've learned over the last couple decades.

Summary

While there are many prioritization frameworks for teams, such as RICE, there aren't nearly enough to help individuals think through their priorities. Now that you've finished this chapter, you have several examples of frameworks that have (and haven't) worked for me and are well-positioned to think of what the right framework might be for you. There's no one solution, but you're going to accomplish less in your career if you're so focused on correctness that you lose track of keeping yourself energized.

Find links to further reading and resources on *https://lethain.com/eeprimer-refs-9*.

Meetings for an Effective Engineering Organization

Some engineers develop a strong point of view that meetings are a waste of their time. There's good reason for that perspective, as many meetings are quite bad, but it's also a bit myopic: meetings can also be an exceptionally valuable part of a well-run organization. If you're getting feedback that any given meeting isn't helpful, then iterate on it and consider pausing it for the time being. It may not be useful for your organization yet, but don't give up on the idea of meetings.

Table 10-1 shows a representative monthly calendar for an Engineering organization, with four organizational meetings each week. Many organizations use a schedule along these lines for teams of 20 to 200 engineers. Smaller organizations often collapse all these meetings into their weekly team meeting, and larger organizations typically shard these meetings across focus areas (e.g., product engineers within each business line, data engineering, infrastructure engineering).

Table 10-1. Representative monthly calendar for Engineering organizations

Mo	Tu	We	Th	Fr
Team Meeting	Eng Q&A Monthly	Tech Spec Review	Incident Review	
Team Meeting	Eng Mgrs Monthly	Tech Spec Review	Incident Review	
Team Meeting	Staff Eng Monthly	Tech Spec Review	Incident Review	
Team Meeting		Tech Spec Review	Incident Review	

In this chapter, we'll dig into:

- Why you need meetings, even if you've learned to distrust them
- How to run an effective weekly team meeting
- How to maintain your organization's technical quality with tech spec reviews and incident reviews
- Who should attend monthly Engineering manager and staff engineer sessions
- Why a monthly Engineering Q&A session is my favorite meeting
- How to run and scale these meetings as your organization grows

Even if you don't care for most meetings, I hope that you'll come away agreeing that good meetings are the heartbeat for your organization.

Why Have Meetings?

At many companies, the richest vein of communication can be found in the informal relationships between long-tenured employees. The clearest project update is tracked in a ticket tracker. Your Engineering strategy (discussed in Chapter 3) is best communicated in a written document. None of these ideal approaches resemble a large meeting, which isn't too surprising: large meetings are rarely the best communication solution for any particular goal. However, they are a remarkably effective backup solution when there are gaps in your default approaches.

At a large company, you can't discuss everything with every member of your organization, but you can have each manager talk with their team and complement that with an organization-wide meeting. You can't include everyone's input early in every technology decision, but you can ensure that the decisions are well-vetted by sharing them in a tech spec review. While meetings are rarely the best initial solution, they are often the best backup solution to ensure important information is communicated across the organization.

While most leaders view organization meetings as a necessary mechanism to distribute context down the reporting hierarchy, I find that they are also effective at two other important tasks: communicating culture, and surfacing concerns from the organization. Accomplishing all three goals within your organization may require a bespoke mix of Engineering meetings depending on its size, communication norms, and all-company meetings.

That said, I'd like to recommend six core meetings that I recommend every organization start with, and that I've found can go a surprisingly long way.

Six Essential Meetings

These six meetings are split across three operational meetings (weekly team meetings between each manager and their direct team, tech spec review, incident review), two monthly developmental meetings (staff engineers, Engineering managers) and finally one monthly Engineering Q&A to learn what the organization is really thinking about.

WEEKLY ENGINEERING LEADERSHIP MEETING

One of your key roles as an executive is to keep your team moving together toward the same goals—what's really important right now? Another is setting the team's pace—what have we gotten done this week? Both are best done in a weekly working meeting with your team of direct reports and your key stakeholders.

The Engineering leadership meeting, along with a similar meeting run by the CEO with the company's leadership team, is your core hub for operating the Engineering organization. First and foremost, this meeting is a working session for your leadership team to accomplish things together. If there's something you need to solve (e.g., disagreement on strategy, conflict about a new career ladder), use this session to resolve it.

Second, it's an opportunity for your team to share context with each other instead of just with you. What's going right or wrong for them this week? How could their peers help them with their current project? Serendipity starts with mutual awareness across the team.

Finally, it's your forum for forging your direct reports into a "first team" whose top priority is to support each other. (If you've not read Patrick Lencioni's *The Five Dysfunctions of a Team* (Jossey-Bass), it's a quick, remarkable read on this topic.)

Some suggestions for making your weekly leadership meeting effective:

- My weekly leadership meeting always includes my direct reports and usually includes our key partner in the Recruiting, People, and Finance teams. (I've also experimented with including engineers, which has been a constructive albeit mixed experiment!) Having cross-functional partners in the room builds trust across functions, and will often allow you to resolve issues immediately instead of putting them on the backlog to resolve later.

- Maintain a running agenda in a group-editable document, so anyone in the team can add agenda topics over the course of the week. Take a few minutes before the meeting starts to prioritize topics for the upcoming session.
- I'd strongly push the core Engineering leadership team to meet weekly. I would only reduce that frequency at very small companies (<20 engineers) or slow-moving businesses (when not much changes month to month).

However you decide to run your weekly leadership team meeting, remember that your direct reports are likely to model their meetings on yours. If you run a tight, effective meeting, then it's likely that your entire organization will run similarly effective team meetings.

WEEKLY TECH SPEC REVIEW AND INCIDENT REVIEW

The tech spec review and incident review are weekly sessions for discussing any new technical specs or incidents that have taken place since the previous week. On a quiet week, both of these meetings might be canceled, but on a great week, both will push the company forward in very different ways.

Each company will run a distinct version of these meetings, and the meetings themselves will evolve a bit over time. A few suggestions on what I've seen lead to effective review sessions:

- All reviews should be anchored to a concise, clearly written document.
- Reading the written document should be a prerequisite to providing feedback on it. Some teams find that it's helpful to start with 10 minutes to read the documents. Alternatively, it may be helpful to block the preceding 30 minutes before the review meetings on Engineering calendars for independent reading.
- Good reviews are anchored to feedback from the audience, and discussion between the author and the audience. Bad reviews are anchored to the author presenting their document. You will only have good reviews if you're diligent about steering folks away from presenting their writeups.

- There should be a clearly documented, very simple process for getting your incident writeup or tech spec scheduled. An effective approach I've seen is having a dedicated room in Slack to request that your document is added to the next meeting.
- Long term, you can measure the impact of these review meetings by whether folks are submitting their new tech specs and incidents for review. If the forums are struggling to solicit content, it's generally a sign that you should work to make them more useful to attendees.
- Prepare first-time facilitators with a practice run with an experienced facilitator! This will make them more confident, foster more valuable discussions, and prevent spending thousands of hours listening to folks who mistakenly believe the review should consist of them reading their document out loud.
- Running these reviews well is time consuming, and I highly recommend finding dedicated owners for each meeting. Although a bit annoying at times, both are highly impactful leadership opportunities that can be run by an experienced engineer.

Many Engineering leaders skip these reviews when they get busy—and it's certainly a bad sign when you are the loudest voice in either forum—but they're important forums to be present in. They're rare opportunities to see engineers across many teams and a wide range of seniority interact on relatively high-stakes topics. These meetings are a clear barometer of your Engineering culture. If you're uncomfortable, bored, and frustrated by these meetings within your own company, fix them.

MONTHLIES WITH ENGINEERING MANAGERS AND STAFF ENGINEERS

I run two monthly meetings focused on ongoing team development. The first is the Engineering managers' monthly, attended by all Engineering managers, and the second is the staff engineers' monthly, attended by all Staff-plus engineers (and generally without managers). In most organizations, these groups are in the same order of magnitude, and in all organizations they are equally important to a well-run Engineering organization.

The format varies a bit for each session, but is generally along the lines of:

1. [15 minutes] Ask each member to take one minute to share something they're working on, something they're excited about, and something they're worried about.

2. [30 minutes] Someone, often me, presents on a development topic. The topic can be anything that creates an opportunity for the group to develop their professional craft. For the managers session, it might be something I've recently read like a Lara Hogan piece about delegation or something I've written about like reading a profit and loss statement (the topic of Appendix C). For the staff engineers session, it might be a topic from Tanya Reilly's *The Staff Engineer's Path* or my own *Staff Engineer*.
3. [15 minutes] Close with an open-ended Q&A.

If there's a pressing topic—a recent reorg or some such—then we'll certainly focus on that instead, but I think it's valuable to support the team members in their personal development. Done well, attendees will leave each session with more value to offer the team and more confidence that the company is invested in their success.

MONTHLY ENGINEERING Q&A

The final meeting I recommend is the Engineering Q&A. I run it monthly for an hour, starting with introductions of new team members, and then taking a few minutes to share any messages I'd like the team to think about. Then we move into Q&A for the remainder of the session. If we run out of questions, we end early, but I find that rarely happens.

Many engineers won't get to work with you directly, and they will learn to trust (or distrust) you based on how you respond to the most difficult questions during these Q&As. Similarly, you may not hear their concerns in any other forum. The concerns should bubble up to you through the reporting hierarchy, but sometimes there are communication blockages, and this is one way that information will jump the gap to reach you.

Running a good Q&A depends on consistently getting good questions from the audience. To that effect, there are a handful of techniques that I've found valuable:

- I start every Q&A by saying, "I'm glad to talk about anything, no questions are off limits. If your question is too awkward or private, I'll just avoid answering it. Ask whatever you want." This is partially a joke, but it's also how I run the sessions. Sometimes folks ask angry questions. Sometimes they ask questions that were already answered by an email. I always try to bring an even, positive energy, even when the questions are messy.

- Having a good tool for taking questions goes a very long way. I recommend three core features: to be able to ask questions before the meeting starts, to vote on which questions are most important to answer, and to ask anonymous questions. I would always prefer difficult anonymous questions over folks feeling too uncomfortable to ask them, even if it makes for an awkward session.
- Remind folks the day and hour before the meeting that the Q&A is coming up and steer them toward the Q&A tool to maximize the incoming questions and the number of people asking them.
- Use the Q&A as an opportunity to highlight individuals doing important work or those who are key leaders in your organization. When a question comes in that touches on someone's work, I warn them that I'm going to share my thoughts quickly before passing the question to them, giving them a few minutes to prepare.
- Every organization has a small number of folks who can't help but ask questions. Nurture those folks to ensure you get at least some questions! Good tooling (a question-voting tool) and hygiene (creating pauses even when hands are up to allow others to ask questions, too) will mitigate the potential downside of only answering their questions.
- If your team has a relatively narrow window of time zones (roughly three hours difference), then you can find a fixed time that works for everyone. However, if your time zone window is any larger, consider alternating times between a couple time slots to make sure that everyone is able to easily attend at least every other Q&A. I generally recommend spreading sessions across time zones instead of increasing frequency. I also generally recommend against recording Q&As, as question quality tends to drop. That said, experimentation here is cheap, so try different options if you think they'd work better.

Although you may not immediately agree, the Engineering Q&A is often my favorite meeting of the month. It's where I learn how I'm performing relative to my team and my organization's expectations. Are things going well? Is the team upset about something? Is it a problem I already knew about? Can I fix it to prevent them being upset about it next month? If you're unconvinced about the value of the Q&A, give it a try for a couple months and then go from there.

What About Other Meetings?

Many organizations have meetings in addition to the ones I've proposed, and that's totally reasonable. There are other meetings that I've run and found to be quite helpful, ranging from a weekly session to meet new hires to a quarterly Engineering all-hands meeting where we celebrated launches and spoke about upcoming challenges. Following is a handful of meetings that many companies find useful:

1:1 meetings

I highly recommend weekly 1:1s with your direct reports and the peers you work with most closely. Borrowing from my concrete experience at Calm, during my first year I met with three executive peers every week, and my other peers once a month, for an hour. I met with each member of my team every week for an hour, although as we'd worked together for several years, some weeks those 1:1s happened to be 30-minute meetings instead.

Skip-level meetings

These are very valuable to conduct with your organization, but I've found it difficult to create space for them as an organization grows. My recommendation is to determine a fixed amount of time to devote to skip-levels, say one to two hours a week, and to iterate on frequency and format to stay within that allocation (e.g., do group skip-level meetings, meet once a quarter instead of once a month). Skip-levels feel productive and you'll probably feel bad if you don't do them as often as you'd like, but don't let them get in the way of performing your core work.

Execution meetings

You absolutely want some kind of weekly or monthly execution review meeting, to track and debug execution. The name of this meeting varies tremendously across companies, from Business Review to Operational Check-In to Plan Review.

I've not included this as an essential Engineering meeting because my experience is that this meeting needs to be cross-functional rather than Engineering-specific and is generally run by a function outside of Engineering (often by Product or Business Operations). Any function-specific execution review is likely to assign responsibility to other functions not in the room, whereas a cross-functional review is more likely to resolve issues.

Show-and-tells or demos
: These are culture-forming meetings that make it clear to attendees what kind of work is celebrated at the company. These are often particularly joyful meetings, centering on folks sharing what they're proud of. Frequency across companies varies a bit from weekly to monthly.

Tech talks or lunch-and-learns
: These provide an opportunity for members of the team to learn together, either by teaching areas of expertise or by reading books and papers together.

Engineering all-hands
: You want Engineering to spend time together, but exactly how you want to do this will vary quite a bit depending on the stage your company is in and what other company meetings you have. The other issue is that all-hands meetings are a bit open-ended. Is your all-hands really a show-and-tell? Or a roadmap review? Or something else entirely?

My experience is that most of these additional meetings depend significantly on your broader company meeting calendar. For example, does your company already have a monthly all-hands, and what do you talk about there? Running a monthly Engineering all-hands when you already have a monthly company all-hands is often a bit painful to coordinate, which you'll have to decide about for yourself. I prefer aligning with existing company meetings whenever possible. As important as it is for Engineering to know what other engineers are doing, it's even more valuable for them to know what the folks in other functions are focused on.

Who Runs the Meetings?

As head of Engineering, you'll generally lead your team meetings, manager and staff engineer monthlies, and Engineering Q&As. You may also lead the tech spec review and incident review, but it's likely that one of your direct reports or a staff engineer will take on leading those forums as your organization grows (because they take a level of persistent attention that can be difficult to sustain as an executive focused on putting out fires). If you introduce further meetings like a show-and-tell, the host will vary.

There are three important points to consider here:

- Leading a meeting doesn't necessarily mean that you need to do the coordination work to make it a success. You'll often partner with an executive assistant (more on this in Chapter 16) or program manager on those aspects.
- The team will watch you closely and take cues from your behavior. Even if you say a meeting is important, they'll only believe you if you show up regularly. It's very difficult for others to lead organizational meetings if you don't attend.
- Even if you don't lead or coordinate one or more of these forums, you are accountable for making it a good use of the organization's time. That is something you can't delegate.

As with all things, there are no firm rules about who does what across organizations. If you want to try something different, then give it a try.

Scaling Meetings

Many organizations have more meetings than this, and that's totally fine. Do what works well for you. Similarly, an organization with five engineers would only need one of these meetings—the weekly meeting with their direct team. Scaling down is relatively easy—drop meetings that aren't helpful—but scaling up can get a bit messy.

In terms of scaling meetings, I generally recommend:

- Scale operational meetings to optimize for the right participants. If your tech spec review sessions are skewing toward mobile concerns, but most attendees don't have context on mobile, then you may want to split out a dedicated mobile session.
- Scale developmental meetings to optimize for participant engagement. If you have more than 10 folks in a developmental meeting, many of them will stop participating or even stop attending. If you have more than 20 Engineering managers, consider pushing this meeting one layer down the reporting hierarchy, such that your reports run their own development meeting with their managers rather than you running it with the entire organization. (You can, of course, find hybrid approaches! You could run the meeting every other time. You can focus on splitting into small discussion groups after introducing the content, and so on.)

- Keep the Engineering Q&A as a whole organization affair, but consider having your direct reports roll out their own Q&As as well at a certain size (e.g., an Engineering organization with 600 engineers can certainly tolerate an Engineering Q&A and also a Product Engineering Q&A every month).

Finally, I find that many organizations split important meetings prematurely because there are one or two individuals who behave badly in them. For example, you may have an antagonistic engineer who joins every tech spec review meeting to advocate for a particular technology. It's extremely valuable to solve that problem by holding that engineer accountable to a higher communication standard. You cannot scale large meetings without holding participants responsible for their behavior.

Summary

This chapter has covered the meetings you need to have to operate an Engineering organization. This template will help you diagnose missing meetings within your organization, whether it's missing only one meeting, or whether it doesn't have any meetings at all. If you're joining an existing organization, your best bet is to integrate these meetings with the wider company's approach, rather than wholly overwriting what already exists.

Remember that you can't communicate effectively solely through meetings, especially as your organization expands across more time zones. Great communication includes meetings but requires strong individual relationships and working through many other channels (e.g., email, chat), but all of these are easier to create on top of effective meetings.

Find links to further reading and resources on *https://lethain.com/eeprimer-refs-10*.

Internal Communications

Whenever an executive joins a new company, there is an awkward merger between the executive's preferred communication style and the norms that the organization has already established. I remember one executive complaining that engineers weren't reading emails at his new company. He "solved" that problem by sending another email, this one instructing the team that they were responsible for reading their email twice a day. You won't be shocked to learn that this didn't really solve the problem.

This chapter covers five practices that will significantly improve the quality of your organization's internal communication:

Maintain the drip
: Communicate on a regular basis, even if you don't have something novel to share.

Test before broadcasting
: Review your communication with a few people before sending it, to improve clarity and avoid blind spots.

Build the packet
: Ensure every communication has a concise summary, and link to background context and an explicit way to ask questions.

Keep it short
: Edit messages for brevity.

Use every channel
: Distribute important information across every relevant channel, including meetings, email, chat, and so on.

Importantly, these are all mechanical practices. Even if internal communications is something you've struggled with in the past, there are no prerequisites to adopting these tools. They don't require unique literary flair, charismatic public speaking skills, or anything extraordinary beyond the willingness to implement them consistently.

Maintain the Drip

Large companies often make their executives seem larger-than-life, but you're just a person and you can close the distance between yourself and the team by sending a short, weekly email. This is a good opportunity to let folks know what you're focused on and what you're doing, and to reassure them that you're paying attention to their work.

The format here is highly variable: I've iterated on my format quite a few times, and you'll find a format that works well for you. Over the course of each week, I would accumulate my weekly updates in one document, then spend about 20 minutes compiling them into the final email. I'd generally email the organization I lead, along with an email group that anyone internally can join, allowing interested folks outside of my organization to also read the updates. (Personally, I'd recommend pre-populating that group with folks who you want to read the update, including those you work closely with on the executive team.)

A typical weekly update is generally structured as follows:

- One to two sentences about something human. Something that surprised me this week, something that energized me this week, something that made this week stand out for me.
- One sentence summarizing any key reminders for upcoming deadlines or dates.
- One paragraph for each important topic that has come up over the course of the week. These might be a quarterly planning update, a controversial tech spec, a product launch, a partner escalation, or any other topic that feels important. I usually pick two to three topics for each week.
- A bulleted list of brief updates. These might be an incident review, a tech spec, a product design, an interesting discussion in chat, or anything else I want to create some visibility around.
- Close with an invitation for folks to reach out with questions, thoughts, and concerns.

It's a simple template, but I've found it surprisingly valuable for staying aligned. It has also generated a surprising number of unexpected emails from the team, surfacing topics that I wasn't thinking about at all (and generally should have been thinking about more).

These updates establish a persistent drip of information from you to the wider organization. Some weeks you might have a light update, but rather than folks thinking that you're not doing much, they'll be reassured that they are up-to-date on what's important. Dense updates are good, but it's the brief updates that reassure the team that you would have updated them if there was something they needed to know.

Extract the Kernel

As someone who's served as an executive for a while, I've started to notice recurring communication challenges between executives and the folks they work with. The most frequent issue I see is when a literal communicator insists on engaging in the details with a less literal executive. I call the remedy "extracting the kernel."

For example, imagine a team is presenting about their upcoming timeline, and the CEO asks, "Can't you just use ChatGPT to solve that instead of building a custom model?" From there, the conversation will often derail into a debate about ChatGPT versus building a custom model, but the CEO's point is almost always not about ChatGPT. Instead, their point is that the timeline feels too slow. If the team anchors to responding to the specific suggestion, they'll miss the more important discussion entirely. Even if they convince the CEO that a custom model is the better choice, the CEO will be annoyed that their real concern about the schedule wasn't addressed.

The team presenting to the CEO could rightly argue that executives ought to be better communicators. That's certainly true, but the reality is that executives are human. You'll make much more progress by focusing on improving how you communicate with them than by blaming them for their deficiencies.

I recommend that teams receiving executive feedback try to "extract the kernel." When you get a question from an executive, focus on understanding the insight or perspective *within* the question. Then confirm that insight with the executive explicitly. Going back to the ChatGPT example after the CEO recommends using ChatGPT, you could confirm back by saying, "To make sure I'm understanding, the most important feedback here is that we should figure out how to accelerate our timeline?" Now you can ensure that the CEO's feedback is addressed rather than getting caught up on the incidental details in their question.

Test Before Broadcasting

A surprising amount of executive time is spent cleaning up messes. Some of these messes are externally motivated, like a competitor launching a new product, but a surprising number are self-inflicted. At least half of the self-inflicted messes that I've seen executives create are caused by unsympathetic or confusing communication, which you can easily prevent by testing your communication before widely broadcasting.

This may not be necessary when you directly know every member of the team, but by the time there are four or five other managers in your organization—around when you reach 40 folks—at least one other person should proofread every significant organizational communication before you share it out. By "significant," I mean a change, decision, or something the team might feel strongly about; by "organizational communication," I mean something that impacts a wide number of folks, and is almost always beyond the scope of just one team.

The process here is pretty straightforward: write your communication and then ask for feedback before sharing it widely. I recommend directly asking individuals for feedback, as that tends to inspire more useful feedback than asking a group (e.g., asking your full team of direct reports and a few cross-functional peers), but experiment with different approaches until you find something that works for you. The request should be short and direct, along these lines, "Hey Mary, I'd really appreciate your feedback on the performance review announcement before tomorrow at 9AM PT." Depending on the feedback you're looking for, you might focus your request for feedback on "unclear areas" or "anything unexpectedly controversial."

Some folks have a very negative reaction to the idea of getting their communications reviewed. I was initially frustrated by it as well! It can feel like "wasting" time. What I've come to appreciate is that testing the message speeds up the pace of communication, because it reduces the time spent explaining the message. It feels slow, but testing your messaging is better for both your recipients and for you.

Build the Packet

Early on with a small team, communication with your organization will be informal. However, as the team gets larger, a bit of consistent structure will go a long way. I strongly recommend structuring every piece of important communication as follows:

Summary

Brevity is always best, but longer communications are tolerable if they begin with a concise summary that answers the key questions: what's changed? who does it apply to? what do I need to do if it applies to me? where can I find more information? when is it happening? what should I do if I have questions?

For example: "tl;dr – Starting tomorrow (2023/1/20), any proposal for a new service must be made in a tech spec and presented at a tech spec review. See go/tech-spec for more details, and ask questions in #tech-spec or reach out to Jenny X."

Canonical source of information

Important announcements will be shared in many places, but you don't want folks to assume each notification is the most up-to-date, canonical place to check for information. Link to the best place for more in-depth information, generally a document or wiki.

Where to ask questions

As you communicate to more folks, inevitably there will be edge cases that your announcement misses. For example, the previous announcement about new services needing to be reviewed at a tech spec review might be missing information about applying that decision to a newly acquired business unit that doesn't currently work with your tech spec review process.

The best place to ask questions is usually a chat channel or email list, along with a named individual for folks who are uncomfortable asking in

public (typically newer hires or less senior folks who have good questions but are still getting a feel for how communication works).

Earlier, I mentioned the idea of mechanical practices, and building the packet is a particularly good example of a mechanical practice. There's no artistry here, but consistently following this practice will make it easier to land important communications.

Keep It Short

Some people look at a five-paragraph note and say, "I'll read that later." Others say, "I'm never going to read that." Either way, if you write a long message, only a few people are going to read it. The packet format provides structure for keeping your notes short, but you still have to be willing to edit for brevity.

Use Every Channel

Over time, I've developed a theory of organizational communication that I call, "information herd immunity." It's simply impossible to ensure every member of a 2,000-person Engineering organization knows everything. Even if you personally tell each member of the organization that performance reviews are starting next week, and send them an email the day before, some fraction will still be utterly confident that no one told them about performance reviews starting.

Accepting that reality, the secret to successful communication is ensuring that someone on every team knows important information. The best way to do that is to communicate across every channel, and to even create new channels that are currently missing.

Although the specific communication channels will vary a bit depending on the specific topic, here are some of the communication channels I recommend using:

Email and chat

Most companies rely on either email or chat as their primary communication vertical but even in a company that predominantly uses chat, you'll find some individuals who rely more on email and vice versa. Everything important should be communicated across both.

Meetings

These are an important communication channel (and discussed in more detail in Chapter 10). Reiterate important topics in large meetings like a monthly all-hands, and also make sure to discuss the same topics in

your weekly team meeting. It's particularly valuable to establish a communication waterfall from the executive team weekly, to your Engineering leadership weekly, to your direct reports' weeklies with their teams, and so on.

Meeting minutes

Most recurring meetings should generate minutes that document new information and resulting decisions. Sharing these minutes widely is a particularly valuable way to keep more folks aware without them feeling obligated to attend meetings where they might not participate.

Weekly notes

As mentioned earlier in the chapter, it's particularly valuable for executives to share weekly notes with their organization. I've found that only a subset of folks read these notes, but those who do tend to be loyal readers who propagate the information across their teams.

Decision log

It's very helpful to maintain a decision log of recent decisions that a given product, team, or organization has made. These are particularly helpful for acclimating new members of the team with the status quo and how you got here.

Most importantly, communicate across every relevant channel. Any given individual will prefer one or two, and you'll reach the full team by using all of them.

Summary

This chapter presented good internal communication as a series of habits, and recommended five practices that anchor effective internal communication. Whether you rate your organization's communication highly or as non-existent, you will make quick improvements by maintaining the communication drip, testing your communication before broadcasting it, consistently building information packets with the necessary information, and communicating across every internal channel.

Find links to further reading and resources on *https://lethain.com/eeprimer-refs-11*.

Building Personal and Organizational Prestige

Most months I get at least one email from an Engineering leader who believes they'd be a candidate for significantly more desirable roles if their personal brand were just better known. Similarly, when funding is readily available during periods of tech industry expansion, many companies believe they are principally constrained by their hiring velocity—if their Engineering organization's brand was just a bit better, they believe they'd be hiring much faster.

Writing a great deal online, and working at companies during periods when they invested heavily in their external Engineering brand, I've seen both the opportunities and the limitations of personal and organizational brand building. From those experiences, I've come to believe that the value of brand building is overemphasized, but that the impact of enhancing prestige is underrated, and that there's a relatively straightforward playbook to increase your personal and organizational prestige.

In this chapter, we'll work through building prestige, covering:

- The distinctions between building prestige, building a brand, and building an audience
- Deciding whether it's valuable to build your personal and Engineering prestige
- The playbook for manufacturing prestige with a small quantity of high-quality content
- The pitfalls of measuring prestige, and what to measure instead

After reading, you'll be equipped to decide whether to invest in these areas for yourself or your Engineering organization, and you'll have a clear path forward.

Brand Versus Prestige

Your *brand* is a deliberately crafted, sustained narrative that is widely known about you. You don't have to research Google Engineering to have an opinion about Google Engineering. In your career and as an Engineering leader, you will likely be given the advice that it's *very important* to build a brand.

If you participate frequently in social media, it's easy to get sucked into its reality distortion field. When you spend a lot of time in a given online community, being well-known in that community feels equivalent to professional credibility. However, my experience is that few of the most successful folks I know are well-known online, and many of the most successful folks I know don't create content online at all. Maybe they have an Instagram account, but it focuses on their family and non-professional interests.

Enough folks find this counter-intuitive that I'll emphasize this theme by expressing two points:

- The majority of successful executives I've worked with don't write online. They don't post on Twitter or Mastodon. They haven't written a book. They don't speak at conferences. They don't have a YouTube channel. They don't stream on Twitch. In your Engineering leadership career, you will at times be immersed in the message that you need to be creating content to be successful, but there's abundant evidence to the contrary. You absolutely don't have to do this sort of thing.
- Similarly, most Engineering organizations spend little time developing their external brand, and are not externally well-known. For every Meta Engineering blog or Netflix Engineering blog, you'll find hundreds of Engineering organizations with limited public visibility around their work. Many of those silent organizations are doing very interesting work; they just don't spend much time talking about it publicly. You can, without a doubt, be a successful Engineering organization without ever doing any external communication to build your brand.

The *prestige* is the passive-awareness counterpart to your brand. Rather than being what someone already knows about you, it's what someone can easily discover about you if they do some searching. Many interviewers won't know anything about me, but a few minutes of research will find my writing, conference talks, and work history.

You can build prestige in a couple ways:

As an individual

: By attending a well-respected university, joining a well-known company, or giving a recorded conference. More hiring managers will be interested in hiring you if they can watch you give a conference talk online, or if you worked at Google rather than a 10-person startup (even if that startup did amazing things).

As a company

: By focusing on a problem that's immediately attractive to software engineers, finding an attractive way to approach that problem, or retaining prestigious employees. More engineers are interested in working on self-driving cars than on automating personal taxes. If your company does work on automating personal taxes, engineers would certainly be more interested in fully automating that process than streamlining back office processes for a team of accountants.

While many successful Engineering leaders and Engineering organizations don't have much of a brand, most are prestigious in one way or another. Prestige is a universal lubricant. It opens the door to taking senior roles and recruiting senior candidates. It creates edges in your network graph that open doors across the industry.

Thinking About External Communication

Sometimes you'll hear an argument that having a personal brand is important for Engineering executives because they play an important role in external communication, but in practice that's fairly uncommon. Few companies lean heavily on their Engineering executives for external communication. When this does happen, it's generally done by a founding CTO, such as Honeycomb's Charity Majors, who has been very successful in using her considerable reach to create visibility for the

company. You'll rarely see an external CTO hired to take on that sort of role, and there are even fewer examples of individuals succeeding as an external CTO hired into such a role.

After raising their Series A or Series B funding, most companies bring in specialists to lead external communication roles. If they're trying to generate leads for a Sales team to process, then they'll likely want Marketing to lead that. If they're trying to increase product sign ups, then they'll likely want either Developer Relations or Growth teams to own that process instead.

Is Building Prestige Worthwhile for You?

Here are some questions to consider when evaluating whether to invest more time into building prestige:

- Are you able to hire senior candidates to work in your organization? (Particularly those with more applicable experience than you.)
- Does your team seek you out for career advice and advice beyond the immediate scope of their current work? (Not just your direct reports, but more widely.)
- Are you able to start the interview process for jobs you're interested in? (Not necessarily receive an offer, but start the process.)
- Is an executive recruiter able to match you with interesting roles? (Particularly roles that are more complex or desirable than your current role.)
- Is your network expanding by default, allowing you to reach out further and to more senior individuals? (Prestige expands its reach by default, as those you already know go into more senior roles.)

If you answer "yes" to most of those questions, then I wouldn't invest much additional energy here. On the other hand, if you answered "no" to many of those questions, if you didn't attend a prestigious university, work at a prestigious company, or select a core business space that software engineers are interested in, then it's worthwhile to learn how to manufacture prestige.

Manufacture Prestige with Infrequent, High-Quality Content

In my experience, engineers confronted with a new problem often leap to creating a system to solve that problem rather than addressing it directly. I've found this to be particularly true when engineers approach a problem domain they don't yet understand well, such building prestige.

For example, when an organization decides to invest in its Engineering brand, the initial plan will often focus on project execution. It'll include a goal for publishing frequency (e.g., publishing to a newly made Engineering blog), for how to ensure content is representationally accurate across different Engineering sub-domains, and for how to incentivize participants to contribute. If you follow the project plan carefully, you will technically have built an Engineering brand, but my experience is that it'll be both more work and less effective than a less systematic approach.

Prestige is an ambient, positive familiarity. This doesn't require an organizational program or a strict content calendar; rather, it depends on building awareness of a small amount of noteworthy accomplishments. The most effective approach I've seen is doing a small amount of writing or public speaking, and then ensuring that work is discoverable.

The steps to that approach are:

1. Identify a timeless topic where you have a meaningful and atypical perspective. You're looking for topics in which your writing can remain relevant for decades, and perspectives that demonstrate depth rather than chase controversy. Be mindful that atypical doesn't mean controversial; it usually means introducing an additional layer of detail to a low-nuance debate.

   For some example topics, consider "Productivity in the age of hypergrowth", where I argue that effective onboarding is the key constraint in hypergrowth companies rather than hiring. Or in "Migrations", where I argue that the value of technical platforms should be foremost evaluated through migration cost rather than the capabilities of the technical platform.

2. Pick a format that feels the most comfortable for you. Typically this is either a blog post or a conference talk. The ideal format is one that you're excited about, one that allows you to iterate on the content until it's ready for release, is small enough that you can iterate quickly, and generates a permanent digital artifact (e.g., a video or piece of writing).

   I recommend avoiding books and podcasts in this step. Podcasts are hard to iterate on, as you generally get one take plus whatever you can fix in editing. Books are a difficult format to learn in, as iterating on the content can consume a great deal of time.

3. Once you've picked a format, create the content! Go into this process assuming that you will throw away two or three drafts. Get early feedback, and get that feedback from folks who are experienced in that format: everyone has an opinion on what good content looks like, but not all opinions are equally valuable.

   Your content is done when readers can accurately identify your key insight and enjoy it enough to read or listen to its entirety. It's a good sign if some readers disagree with you: anything interesting will generate some disagreement.

4. Develop an explicit distribution plan for sharing your content. The simplest effective distribution plan is to ask a few friends to share your article online: ten people committing to share your article online around the same time is a fine distribution plan.

5. Make it easy for interested parties to discover your future writing by collecting them on a personal website and through your various presences online (e.g., LinkedIn).

6. Repeat this process two or three times over the next several years.

7. You're done! Many more folks will have an ambient, positive awareness of you, and whenever you interview for a role or show up as a hiring manager, it will be easy for others to discover this earlier content that reflects positively on you. If you liked the process, you can do more, but you don't need to spend more time on this unless you particularly enjoy it.

This approach works equally well for building your company's Engineering brand as it does for building your personal brand as an executive. In both cases, a small amount of positive, thoughtful content will go further than a larger

volume of lesser content, and the short-term distribution benefits of engaging in controversy is at odds with your goal of building prestige.

If this advice feels counterintuitive—if it feels too easy—it's likely because you're applying advice for building a brand or an audience to the rather different topic of building prestige. To build a brand that you measure through an audience, consistency is valuable and volume *does* matter. Prestige doesn't need all that; just easily discoverable content that paints a positive connotation.

Does Anyone Follow This Advice?

Viewing myself through this lens, I've written hundreds of posts but probably only about four have generated significant prestige: "Migrations", "A forty-year career", "Sizing engineering teams", and "Productivity in the age of hypergrowth". I'd very likely be equally prestigious if I'd simply written those four without the surrounding 600.

Going beyond my writing, examples of others who've created significant prestige from a narrow slice of their visible work:

- Charity Major's "The Engineer/Manager Pendulum"
- Tanya Reilly's "Being Glue"
- Patrick McKenzie's "Salary Negotiation"
- Julia Grace's "Scaling yourself during hypergrowth"
- John Ousterhout's *A Philosophy of Software Design* (Yaknyam Press)

Certainly, some of these folks have written or spoken widely but they could have accomplished their prestige-related goals without doing so.

Measuring Prestige Is a Minefield

All corporate initiatives demand a metric to measure their outcomes, which brings us to the messy topic of measuring prestige. I recommend making a small, timeboxed investment, tracking how often content comes up in hiring processes (both as a candidate and as a hirer), and avoiding spending any more time on measurement.

There are several measures I specifically recommend avoiding:

Pageviews

These are usually the easiest measure to instrument, but they establish the wrong incentives. This is because it's much easier to drive pageviews by being controversial than by being thoughtful, but controversy reduces prestige rather than enhancing it. Pageviews also incentivize selecting large audiences ("early-career software engineers") over influential audiences ("technology executives"), whereas the influential audiences will become increasingly important to your career and hiring priorities as you get further into your career.

Social media followers

These are a good measure of reach. Reach is part of distribution and distribution is an important part of building prestige, so this is a useful measure but again suffers from the same issues of incentives as measuring pageviews. There are many ways to build social media reach that are at odds with building prestige, particularly anchoring to controversy, which makes this a poor measure.

Sales

If you're selling your content as a book or course, then measuring sales is attractive. Unfortunately, once again the correlation with prestige is questionable. Particularly if this is your only content, it's likely that selling it will reduce your reach.

Volume

The volume of your writing is also tempting to measure, but this will orient you toward building an *audience* rather than building prestige. There's certainly nothing wrong with writing a lot, but it's an inefficient route toward prestige.

If you invest heavily in your brand, it's unavoidable that you'll end up measuring the aforementioned items, but it's very unlikely that these measurements will be useful to you. You can fight that by looking for better measurables, but I've found it's much easier to address that friction by limiting yourself to a modest investment and channeling the saved energy toward something more directly useful to your business or career.

Summary

In this chapter, you learned how to build prestige for yourself as an executive, and your Engineering organization as an employer. You also spent time on the distinction between building prestige and building a brand or audience, and why it's more efficient to prioritize prestige unless you're selling directly to software engineers. You further spent time on why most measurements will steer you away from your goal, which should encourage you to timebox your investment rather than invest more heavily into measurement.

If you only take one idea away, it's that you should be lightly skeptical of following advice and patterns from the companies and executives around you. Companies and individuals play many different games when they create and distribute content online, and the rules and values conflict across many of those games. Make sure you know what game you're playing, and why you're playing it.

Find links to further reading and resources on *https://lethain.com/eeprimer-refs-12*.

Working with Your CEO, Peers, and Engineering

There are so many stories about hiring a new executive who comes in and wreaks havoc. I've seen Engineering leaders who start with a giant, doomed migration, marketing leaders who accelerate expenses until they necessitate a round of layoffs, and a number of executives who are fired in their first month. When people tell these stories, they're almost always framed as the failure of the individual executives, but these scenarios happen so frequently that I believe there's an underlying structural challenge in addition to individual missteps. Fortunately, the structural trap that snags many executives can be avoided by acknowledging the built-in difficulties and navigating them with a deliberate approach.

When a new Engineering executive is hired, they are usually brought in by a CEO who believes that the Engineering function is underperforming. When you talk to members of Engineering, they will likely have a different perspective, potentially that the CEO keeps changing direction too frequently. When you talk to peer executives and the board of directors, you will hear a third and a fourth narrative about what's happening. Where new executives fail is that they think of these as opposing perspectives, when in reality they are all incomplete but valid slices of a complex situation. Successful executives debug these mismatched concerns until they merge into a single cohesive perspective. Ineffective executives anchor to one or two perspectives and dismiss the rest.

In this chapter, we'll discuss the following topics related to building effective relationships that overcome the core structural challenge of being a new executive:

- Understanding whether you're supported, tolerated, or resented
- Navigating the implicit power dynamics
- Bridging narratives across the CEO, board, peers, and your function
- Avoiding anchoring to previous experience
- Fostering an alignment habit
- Focusing on a small number of changes at a given time
- Embracing transient conflict and preventing persistent conflict
- Surviving a panicking peer executive

After working through these topics, you'll have a clear roadmap for debugging the mixed messages inherent to operating as an executive, and an approach for building durable, healthy relationships that support you through your entire tenure.

Are You Supported, Tolerated, or Resented?

It may sound obvious, but the most effective executives are proactively supported by their peers, the CEO, and their functional team. This is not the common scenario, though. Most members of an executive team operate in isolation within their functional spheres. This is very different from environments in which some functions maintain a long-running animosity toward one another.

There's no website where you can check the status of your relationship with other parts of your company, but it's reasonably straightforward to figure out:

Supported

: This is when others proactively go out of their way to make your efforts successful. For example, if you push for a monolith-to-services rewrite, they'd come to you with concrete concerns about why this might not work out, and ideas for how to adjust your approach.

Tolerated

This is when others are indifferent to your work. For example, they hear about your services rewrite, and pull any related work tickets into their sprints, but don't make an effort to flag structural inefficiencies in your approach.

Resented

This is when others view your requests as a distraction from their actual work. For example, they'll advise their teams to ignore your migration tickets until you escalate loudly.

If you're already supported by most executives and functions you work with, then you're in an excellent place and can move on to the next chapter. If you're not sure, or suspect you're tolerated or resented by some functions, then read on to unpack the most frequent causes of damaged relationships and how to resolve them.

Navigating the Implicit Power Dynamics

Even if you're an executive who prioritizes supporting your team, it's very hard not to come into your new job anchored to your CEO's evaluation of how Engineering is performing. This is natural because you've just spent the entire interview process speaking with the CEO, and the CEO is your boss: if you don't impress them, you'll get fired. However, CEOs hire new executives when there is a problem to solve, which almost always means they've been telling you all about Engineering's problems, but probably not about everything that's going *well*, and their diagnosis of the problem may have some gaps.

You should listen closely to the CEO, the board of directors, and your executive peers' perspectives, but don't close your mind to other new perspectives. Just because Engineering is having problems doesn't mean Engineering *is* the problem, and you'll want to dig in with the team directly before switching from listening to problem solving. To be fair, Engineering is almost always part of the problem, but they are rarely the entirety.

Executives who underestimate the impact of these power dynamics are prone to treating the top-level symptoms as if they are the underlying causes, which leads to superficial and often misguided solutions. When you look at all the executives who quickly mandate widespread migrations or initiate reorganizations, underneath the decision is almost always the sweet siren's song of power dynamics that leads them to dismiss the input of the people doing the actual work.

While I certainly don't recommend ignoring your CEO—they are indeed your boss—you'll serve them more effectively by listening widely before you solve their problem. Moving with too much urgency to address symptoms rarely gets you closer to solving the problem that your CEO truly cares about.

Bridging Narratives

The most talented executives I've worked with have the remarkable ability to weave four or five seemingly incompatible perspectives into a unified understanding. For example, Product might feel that Engineering is slipping on delivery. Engineering might feel that Design keeps changing requirements. Design might feel that Product is making late but reasonable change requests, but that Engineering is intolerant of their change requests. Sales might believe that Engineering is missing all their committed dates because they're not customer-oriented. The simplest explanation here is that Engineering is struggling to execute effectively, but there's undoubtedly a more nuanced diagnosis that goes beyond casting blame, and that nuance will make it much easier to identify an effective solution.

When you run into a complex problem, slow down and bring many different perspectives into consideration before you push to solve the problem. When you get good at this, it doesn't take much more time, and rather speeds up problem solving by skipping the contentious debate around assigning blame.

Executives who practice the skill of bridging narratives are acting as company-first problem solvers rather than operating from a place of loyalty to the function they lead. This leadership creates a culture—and expectation—of cross-functional partnership and is the most effective way that I've personally found to build supportive relationships across functions and executive peers.

Don't Anchor to Previous Experience

Many new executives accidentally alienate their peers and functions by assuming their new company will work the same way that their old company worked. This is common when senior leaders at large companies transition into smaller startups. Those leaders are conditioned to an unnatural degree of support and resources, which simply doesn't exist in smaller companies. Simply by running their familiar executive playbook, they can easily alienate their new colleagues.

Moving from small to large companies brings frequent challenges, as well. New executives often assume they can make major decisions without much

discussion and end up frustrating peers who expected to be included in the decision-making process.

Avoid this by watching how others solve problems in your company and then asking them why they solved them that way. Some executives think it's faster to learn by running into walls (e.g., do what you want and see what breaks), but I've found that to be ineffective when those so-called walls are in fact your coworkers. No one appreciates being run into, and it creates the lasting impression that you're focused on yourself rather than operating as a member of a collaborative executive team.

Fostering an Alignment Habit

Many executives assume feedback will come to them if they make mistakes. This is eventually true, but you'll get the feedback much later than is useful. You should instead build a habit of inviting feedback and making sure you receive it well.

This might look like reaching out to participants in an executive meeting to ask if there's anything you could have done better or should not have done at all. Much of the time you won't get any meaningful feedback, but occasionally you will. When you do get feedback, even when it isn't particularly helpful, say thank you and ask a follow-up question to better understand the feedback. When possible, ask narrow questions like, "Did I ramble a bit when I shared my perspective about reducing bonuses?" which are much easier to answer than open-ended requests for feedback. Most importantly, strive to reward feedback by actually improving on the things that are mentioned.

You don't need to do this all the time with everyone you work with but make a deliberate effort to occasionally ask everyone you work with closely. You should always be asking for occasional feedback in order to ensure folks have an opening to share.

Focusing on a Small Number of Changes

Each time you come to colleagues with an idea, they refer back to their mental list of the last few ideas you brought them. Did those proposed changes go well, and did your peers end up feeling good about them? Are those changes still ongoing without clear resolution? Did they end inconclusively, or did they end up being viewed as a waste of time?

As long as your previous ideas have been effective, then folks will generally be glad to support your next batch. If they haven't gone well, then they won't.

Consequently, your best bet to retain the support of your team and peers is to focus on delivering a small number of changes with meaningful impact. Don't be the executive who pushes a number of technology migrations before finishing the first one. Don't be the executive who claims to rework the planning process without ever using that new process.

Pick a few things that you're confident will work and deliver them. This advice may sound obvious but it's often ignored, and ignoring it will quickly get you placed into the tolerated or resented buckets.

Having Conflict Is Fine, Unresolved Conflict Is Not

Many executives aim to eliminate conflict in their working relationships, which I believe is subtly the wrong goal. Real companies experience rapid changes as they grow and the markets they operate in shift. If you're experiencing zero conflict day to day, then it's very likely that you are ignoring conflict rather than resolving it.

Conflict isn't inherently negative: experiencing new kinds of conflict is an important sign of growth. The sort of conflict that you want to avoid is unresolved, recurring conflict. An example of healthy conflict is an executive team that disagrees on implementing a return-to-office policy. This is a complex topic with a number of different perspectives, and you want to surface all those perspectives en route to deciding on your company's approach. An example of unhealthy conflict is an executive team that strongly disagrees on the size of relative investment between Research & Development (R&D) and Sales & Marketing (S&M), and that continues arguing without alignment each time they revise the financial plan.

When you detect an unresolved conflict, spend time understanding the different points of view and why people aren't able to agree. Much like the challenge of finding a unified narrative that spans CEO and functional perspectives, there is usually a unified view that acknowledges seemingly opposing perspectives. If you can find it, you can often untangle the conflict. If not, it may be time for your executive team to establish a clear escalation mechanism to explicitly resolve points of conflict.

Finally, keep an open mind on whether you are the person preventing conflict from resolving. Sometimes I've gotten so attached to my own perspective that I couldn't let things go, even when it wasn't particularly important, which always ended up causing more harm than it was worth. Even if you disagree with others, it's still worth letting most stuff go to keep the executive team healthy.

Structured Escalations

People often view escalations with suspicion, grouping them with politics as something negative. However, escalations are a key part of doing your job. Sometimes you and your peers will be given incompatible directions that can only be resolved by someone with more context or authority moving one of those milestones. As an executive, this usually means escalating to your CEO.

Most companies have unstructured, undocumented escalation processes that can make escalating feel awkward. But LinkedIn developed a structured escalation process to encourage escalations that build trust. LinkedIn's process is as follows:

1. If you identify a disagreement, agree with the counterparty (e.g., whoever you disagree with) that you'll resolve it in the next five days.
2. Prioritize time with the counterparty to understand each other's perspective and try to agree on a path forward together.
3. If you cannot, then perform a "clean escalation" where you both escalate together with a shared document, rather than escalating separately and losing important context.
4. Commit to following the direction from whomever you both jointly escalated to.

I've found this process works quite well, as long as you hold folks accountable for following it. Otherwise, it's just another trap for rule followers like any other unfollowed process.

Surviving Peer Panic

Sometimes things go awry even if you build effective relationships within the company. The most frequent version of this is when one of your peers starts to get negative feedback from the CEO, and that peer struggles to incorporate that feedback effectively. This is particularly messy to deal with as a newly hired executive, who is trying to build a relationship with a peer who's more focused on short-term survival than building a long-term relationship with you.

Some examples of this happening include:

- A Product executive is being held accountable for recent launches not driving more product usage, and switches to blaming Engineering for the buggy execution.
- A Sales executive is held accountable for missing quota, and switches to blaming slow Product and Engineering delivery.
- A Marketing executive is held accountable for high customer acquisition costs, and switches to blaming Engineering's delivery of a marketing website.

When this happens, try to find empathy for the individual who is struggling to address the concern and then sit down with them to try to solve it together. Your peers will sometimes struggle and sometimes they will point blame toward you, but ultimately succeeding as an executive team is a group activity. Even if an executive is performing poorly today, it's usually less disruptive for you and the company to improve that executive's execution than to find and hire someone new. Even if you aren't ultimately able to help this particular peer succeed, you'll come away with a clearer understanding of how to support their replacement.

At their worst, these situations get pretty unhealthy, particularly if the CEO has written off an executive but doesn't have a concrete plan for exiting the executive from the business. The longer this goes on, the more uncomfortable it will get. At that point, push your CEO to make the change and insert yourself as a buffer between your team and the panicking executive. Until your CEO cleans up the mess, you can at least minimize the disruptive impact on your team.

Summary

At this point, you have a clear framework for becoming, and remaining, an executive who is supported by your peers, your CEO, and your team. Continue to invest in your relationships. Continue to find the overarching perspective that merges the seemingly incompatible views around you. If you're ever uncertain about what to do, search for the approach that maximizes your impact at the company over the next three years rather than the next three months and do that.

Find links to further reading and resources on *https://lethain.com/eeprimer-refs-13*.

Gelling Your Engineering Leadership Team

One of the first leadership books I read was Patrick Lencioni's *The Five Dysfunctions of a Team* (Jossey-Bass), which introduces the concept of your peers being your "first team" rather than your direct reports. This was a powerful idea for me because it's much harder to be a good teammate to your peers than to your direct reports. While your incentives are usually aligned with the team you manage, it's very common for your incentives to be at odds with your peers' incentives. The head of Marketing may want you to prioritize work that your team believes won't be impactful, or the head of People may want you to be more selective in assigning top-performance designations even when you believe your team has earned them. Those are difficult topics to agree on.

Even if it's easier to align with your direct reports than your peers, it is nonetheless surprisingly difficult to create an Engineering leadership team in which members are aligned with each other. Well-intentioned members will occasionally find themselves at odds with one another, and it's your job as the Engineering executive to establish clear values for the team and to referee conduct that falls outside those values.

In this chapter, we'll discuss:

- Debugging the Engineering leadership team after stepping into a new role
- Gelling your leadership into an effective team
- What to expect from your direct reports in that leadership team
- Diagnosing conflict within your team

By the end, you'll have a framework for forming and operating an effective team.

Debugging and Establishing the Team

When you start a new executive role, one of your most important tasks is developing your point of view on your functional leadership team. This can be tricky because often by the time you are hired, the previous executive is long gone and the Engineering leadership team has been operating without guidance or feedback for some time. As such, early glimpses of your new leadership team members are more likely to reveal them at their worst than their best.

Despite that, you need to answer a few key questions in your first couple months:

Are there members of the team who need to move on immediately?

> These are folks who either have significant behavioral issues or who have lost the confidence of their team and peers. Rather than underperforming, these are individuals actively causing damage. For example, if a member is complaining widely about their role and the company, such that they cannot energize the team they lead, it may be time for them to move on.

Are there broken relationship pairs within your leadership team or among members of your team and their key stakeholders?

> You're looking for individuals who clearly cannot work together, or even communicate successfully with one another despite needing to collaborate. For example, if the Product and Engineering leaders on a business line never interact directly in meetings, and contradict each other in private, you need to solve that quickly.

Does your current organizational structure bring the right leaders into your leadership team?

> Your focus should be on adding representation for teams like Data, Security, or Infrastructure where the absence of leaders prevents important decisions from moving forward quickly, and also on elevating important teams from places where they are being neglected or mismanaged. For example, if your company suffers from a significant lack of a security mindset, and your Security team is nestled deep within Infrastructure where it's managed by someone without direct security experience, you may want to pull Security into your leadership team.

As you spend time meeting with folks one-on-one, attending meetings, and watching the team operate, you must quickly form initial opinions on each of these questions. Once you have a first opinion, spend time trying to disprove it. You've been hired to determine the truth, not coalesce the most popular narrative, and you'll never regret being suspicious of simple narratives.

Once you're confident in your interpretation, it's time to take action. Start by moving out the individuals who won't be part of the team going forward. Each time you change members of your leadership team, you will have to gel that team afresh, so it's important to quickly put together a plausible leadership team. This can feel uncomfortable, but letting an untenable situation linger doesn't help anyone. Making personnel changes quickly lets you shift your time from dealing with interpersonal conflict to instead working through more durably valuable decisions around execution and strategy.

Sometimes the decision to be made won't have a clear answer. For individuals who are struggling but whom you believe have a role to play on your leadership team, have a frank discussion. As a new leader, you have a small window for setting clear expectations, which makes it easier to have those difficult discussions. I particularly want to discourage you from waiting to have these discussions because you're afraid you might lose someone you can't afford to lose. Most leaders have a strong desire to perform, and establishing clear expectations sends the message that you're here to help them perform. That's what good folks want! Anyone who leaves because you set expectations is already too frustrated or burned out for you to help.

Finally, once you've edited the members in the room, you have to decide whether to tweak your organizational structure to bring the right folks into the room. It's not a goal to directly represent every member in your leadership team, but there should be someone who can represent the most important teams with reasonably high granularity. Particularly early on in your tenure, I recommend erring on an overly broad leadership team to ensure a wide set of perspectives are surfaced. You'll have to balance that against the value of stability. If you're just not sure, you can always experiment with bringing folks into your meetings on an interim basis.

Note

When it's possible, it's well worth your time to consider having one or two senior engineers to report to you, as opposed to only managing Engineering managers. An Engineering executive is responsible for blending the technical and managerial perspectives of Engineering, and that's most easily done when both perspectives are represented within your leadership team.

In some cases, this will be difficult to accomplish. You may have too many direct reports already and feel it will be overwhelming to include more. In that case, work to establish a group of senior engineers that you interact with frequently to ensure you're hearing their perspectives. This is frequently an Architecture team or the senior-most engineer from each business unit.

Operating Your Leadership Team

Once you've decided on your initial leadership team's members, you have the larger task of turning those individuals into a team. This hinges on creating the values, structure, and relationships your team needs to be effective. Whereas your initial edits on the leadership team will be quick, gelling and operating this team will require ongoing effort throughout the entirety of your role.

Operating your team effectively means boiling down to four themes:

Define team values.

How do you want this team to make decisions? How much should they focus on enforcing consistent policy versus solving escalations? Your team values should connect with your organizational values, but will likely focus more on behavioral trade-offs in how you do work. The most important values to establish are those that explain navigating conflict within the team (e.g., how should the team navigate the trade-off between delivering new product functionality and supporting the migration necessary to deprecate an unreliable component? Do you want the two leaders to find a compromise themselves, document their disagreement and then escalate, discuss the issue as a whole group, or something else entirely?).

Some teams will write team values down and others will simply live them actively. The important thing is having and enforcing them; whether you'll benefit from writing them down will depend largely on the sorts of people on the team.

Establish team structure.

These are the mechanisms and rituals that the team relies on to operate Engineering. What meetings does this team rely on to broadcast context and resolve conflict? How does information flow from your company's executive team down to this group? Do folks give a weekly update over chat?

Find space to interact as individuals.

Many senior leadership teams become so transactional that they feel impersonal. A transactional collection of individuals can do good work, but a gelled group that knows each other is better positioned to work through the inevitable difficult moments. Your role as the team's manager is to create some unstructured space to interact as informal, unstructured humans. That unstructured space is locally inefficient—it certainly won't generate any action items—but relationships are what's necessary for large groups of humans to work together successfully.

Referee defection from values.

Some leadership teams struggle despite having clear values, structure, and relationships. Usually this can be traced back to one or two members of the team violating the team's values without consequence. Once someone defects, others will become skeptical about whether following the values is necessary for them either. Your role as the executive is to hold members accountable to the values. Mistakes will certainly happen, but tolerating consistent violations is the same outcome as not having team values at all.

As is often the case in leadership, these themes are fairly simple, and the hard part is consistent implementation. There will be some initial setup here, and there will also be ongoing maintenance every single week. Fortunately, you are not doing this alone; you'll need the active participation from the team itself.

Expectations of Team Members

A central part of creating your team's structure is setting expectations for how members participate with the team. Often, individuals on their first leadership team will assume that the team exists to help them do their work, rather than recognize that they lead their organization on behalf of the business' goals. Setting explicit expectations helps folks recognize if they have an inverted view of priorities.

It's particularly useful to set team member expectations on:

Leading their team.

Some companies focus internal leaders on bureaucratic execution, lauding pristine headcount spreadsheets and thoroughly documented quarterly plans. Other companies evaluate their internal leaders primarily on how they dynamically firefight issues as they emerge. Another set of companies anchors heavily to visionary leadership rather than operational concerns. It's essential that your leadership team understands however you and your company will judge them.

Communicating with peers.

Some leaders are naturally relationship-oriented and will build a wide network within the company. Those folks will generally know what their peers are working on and what they're struggling with. Not all leaders act that way, though. Another common archetype is the leader who works heads-down, optimizing their team's operations, mostly unaware of what their peers are doing. If you don't tell your team what you expect, they'll default to their natural state, which probably won't be what you want.

Staying aligned with peers.

For leaders, being aware of their peers' goals is a necessary precursor to resolving conflict, but is certainly not sufficient. Your team members should understand your expectations around how they resolve tension around each other's goals. Should they work to resolve the conflicts directly, and only escalate to you when that is unsuccessful? Should they escalate together, or is it more important to escalate quickly, even if it's a bit messy?

Creating their own leadership team.

Although the leadership team reporting to the functional executive is particularly important, each member will have their own leadership team as well. It is turtles all the way down. It's not enough for you to communicate to your team; you need each leader to communicate with their team, and so on: leaders exist everywhere in high-functioning organizations.

Learning to navigate you, their executive, effectively.

One of the enduring controversies in the Engineering management community is whether "personal READMEs," short documents that describe how to work with you, are a sign of professionalism or self-absorption.

> Regardless of your opinion there, the reality is that your team will spend a significant amount of energy figuring out how to work with you, and you should do what you can to make it easy for them. Are you a stickler for process or very much the opposite? How much risk should they take? Which stakeholders should they prioritize? Let them know!

Each of these is a useful topic to discuss in your weekly team meeting or 1:1 sessions. Start early, engage with individuals who are finding aspects of these difficult to adopt, and review these topics periodically as membership of your leadership team shifts. Your team will eventually pick up your expectations even if you don't state them explicitly, but it's much faster to just tell them what you want from them.

Competition Amongst Peers

Sometimes you'll pull together your leadership team effectively. It works well for a while, but a year later you'll notice something's gone wrong: collaboration has slowly faded into internal competition. There are a few different reasons why this can happen, but ultimately it means that members of your team believe they'll be rewarded more for capturing capacity from each other than creating capacity for the organization overall.

The three most common causes of competition that I've seen are:

- A perceived lack of opportunity
- The application of poor habits learned in the context of shrinking or bureaucratic companies
- The failure of a leader to referee their team

The first of these three, a perceived lack of opportunity, has been particularly common in my experience. If your team believes they've exhausted their personal runway within your team, then they'll look for ways to create opportunity for themselves, which often means competing with peers. It's certainly true that only one person can take on any given stretch assignment.

When you're managing less experienced folks earlier in your career, it makes sense to solve this problem for them by giving them specific opportunities for growth. For senior leaders, I recommend giving the problem back to them: the next step in their career is being an executive, and certainly no one is going to tell them how to advance their scope at that point. Particularly, encourage them to look at opportunities to grow themselves and their career in ways that aren't

zero-sum. With some creativity, there's no reason for your team members to compete over access to opportunity.

The next common challenge is team members who've learned bad habits in bureaucratic or shrinking companies. Those environments often teach the implicit leadership lesson that it's more effective to capture existing capacity within the company than to create new capacity for the company. Leaders immersed in that lesson often view success as a zero-sum game. Your goal is to convince them that today's capacity represents a small fraction of the future capacity. By instead focusing on the capacity that can be collaboratively created in the future, they'll have significantly more opportunity.

Finally, if these first two explanations don't fit particularly well, then it's likely the issue here is you: the team knows you'll tolerate bad behavior. Teams composed of ambitious individuals typically only follow rules when following those rules is the best path forward. If they're defecting from your stated rules, then you're not enforcing them effectively. Get back to the basics that this chapter started with.

Summary

Working with a new team, particularly one that has been trying to lead itself without an Engineering executive for some time, is both messy and necessary. This chapter covered getting started with debugging your new Engineering leadership team and operating that team once it's gelled. Keep in mind that, even if you do a remarkable job with your team, you'll be spending time on this from your first day until your last.

Find links to further reading and resources on *https://lethain.com/eeprimer-refs-14*.

Building Your Network

In most of my roles, I've learned more from my peers than from my manager. Even when you get along well with your manager, your peers' perspective will usually be closer to yours than your manager's. Once you transition into an Engineering executive role, you'll still have peers, but they're a different sort of peer. They will look at problems from a very different perspective than yours. If you ask the head of Product for feedback, they will give it, but it'll come from a Product perspective. This will make your peer executives' feedback valuable, but valuable in a very different way from the peer Engineering leader feedback you'll have gotten in previous roles.

When I started my first Engineering executive role, I spent time building a learning circle of industry peers, which was fundamental to my success. Whenever I got stuck, I was able to quickly poll their perspectives and find new ideas to address my problem. Whether you build a learning community or rely on cold outreach, building your Engineering leadership peer network is one of the most valuable steps you can take as a new Engineering leader and will significantly speed up your rate of learning.

In this chapter, we'll cover:

- How to leverage your network effectively
- The unfortunate truth that there isn't a cheat code to building your network
- How to build your network slowly over time
- Other valuable networks to cultivate outside of engineers

By the end, you'll have a clear plan for getting started, a realistic sense of what's possible, and an understanding of the various tools available to build with.

Leveraging Your Network

Whenever I run into a new problem, I try to solve it myself or work through it internally with my team and peers. If I'm still not pleased with our approach by that point, I reach out to my network. During my time at Calm, I reached out to my peers on a number of topics, but especially when I was figuring out how to measure the Engineering organization and when I was writing Calm's Engineering strategy. In turn, my network also reached out to ask me about Engineering meetings, finding their first executive job, and pretty much every topic you can think of.

Here are a few of the ways that I've leveraged my network over the past several years:

- To collect data points on the size of platform engineering teams at different companies, to better benchmark the size of my own platform engineering team.
- To understand pricing when negotiating renewals for our highest-cost vendors.
- To solicit advice on navigating complex situations within my executive team, which I'd prefer not to discuss with anyone within my company.
- To calibrate job offers when there's a repeated gap between compensation data and candidates. For example, has the market moved or are these candidates getting unusual offers?
- To decide whether to move on from a job, including understanding the state of the job market for executive roles.
- To gather backchannel feedback on senior leadership hires who got mixed feedback during our interview process when we had a short timeframe to extend an offer due to an outstanding offer from another company.

My experience is that no topic is off limits, and people are glad to share almost anything privately. Career ladders, strategy documents, compensation bands—all of these things get shared regularly. If you're having trouble getting the information you need from people, it's probably not because you've broached a sensitive topic; more likely, it's because you're not respecting their time and social capital as much as you could.

What's the Cheat Code?

If you write about peer networking, there are two inevitable outcomes. First, folks ask to use your network instead of building their own. I understand that perspective, but it shows a misunderstanding of how executive networking works: it should be an exchange of value. Asking to take someone's existing network is not an exchange. Most people will ignore that kind of outreach, and over time I've come to ignore it as well.

Second, people will ask for a solution where they can pay to build their network. Sure, they understand that there isn't room in the learning circle I run, but where can they buy their way into another, equally good, learning circle? Unfortunately, I don't believe the incentives of paid networks are particularly good. Paid networks monetize by growing their attendees, which compromises their ability to prioritize high-quality participants. It also complicates finding effective moderators, and giving participants feedback when their actions don't contribute to the group's success.

I highly recommend going into networking with the mindset that it's going to take some work, and that you're going to need to be useful to the folks who will network with you. If you go in looking for a cheat code, you're going to waste your time and the time of the folks you reach out to.

Building the Network

There's no way around it: building your network is difficult, particularly as a newer executive, because you're likely ahead of your peer group in your career. Even if you know hundreds of folks in the industry, only a few of them may have experience as an Engineering executive.

Early on, my networking was very broad. I was excited to meet anyone working in technology. Over time, I've learned to be more intentional about my expansion. At one point, I focused on meeting the infrastructure engineering leaders who were solving the same kind of scalability issues I was. At another point, I focused on meeting people who worked for the vendors I depended on heavily. (It's always helpful to have a friendly escalation point person within a critical vendor organization.) Later, I made it a point to meet folks who've been successful in writing and speaking in the industry.

Whatever it is that I'm learning at any given time—that's what informs my outreach goals. This approach helps me to focus on relationships that are helpful to me, and where I'm willing to make reciprocal investments in the counterparty.

Let's take a look at some specific tactics for connecting with the people you're most interested in.

WORKING TOGETHER

By far the best way to build your network is to spend time working at a large, fast-growing company in a central tech hub like San Francisco, New York, or London. Five years later, your colleagues will have spread across the industry. It's a bit like meeting friends in college: your network growth just happens, without much deliberate effort.

I highly encourage folks pursuing executive roles to spend at least a year or two in this sort of company early-ish in their career. It's not the only way to build out your network, but it is by far the easiest.

COLD OUTREACH

The quickest way to build out your network is to send out quick, concise notes to folks near, but not quite in, your network (e.g., someone you don't know but with whom you have a mutual friend). This works because people want to be helpful, as long as you make it easy. There are a few rules to doing this well:

- Keep it short: aim for two to four sentences.
- Ask a specific question.
- Do not directly ask for time. (However, it is fine to offer time, "Also glad to grab coffee near you if that's easier!")
- Don't ping for a follow-up if you don't get a reply. (Sometimes you won't get a response. It's fine to ask a separate question in a few months, though.)
- If they reply, say thank you and acknowledge their specific response.
- It's fine to use the same email for a few different people.

If you follow this process a few times, particularly if you start your next question with context on how the previous advice worked out, you'll soon be communicating with an acquaintance rather than a stranger. Your network has expanded by one!

COMMUNITY BUILDING

I've benefited a great deal from putting together and running an Engineering executives learning circle. That circle built meaningful relationships across more

than a dozen folks, who I'm able to learn from every few weeks during our sessions, and also able to ping with questions.

What initially made this learning circle work was the willingness to pull together the initial cohort, maintain a code of conduct, and run the group for several years. What's made it continue to work is selecting a group of executives who are thoughtful, strong communicators and interested in learning from each other.

The limiting factor on these kinds of small communities is people willing to do the work to operate them. If you're focused on the idea of joining someone else's community (learning circle or otherwise), instead of running your own, you should have a clear point of view about why you're a very desirable learning circle attendee. Otherwise, you're trying to consume someone else's community and network. This can work, but in the long run you can go so much further, with so much less competition, if you create a net new community rather than relying on consuming existing ones.

WRITING AND SPEAKING

One way to build your network is by writing and speaking in public. My blog, books, and conference talks have all been extremely valuable in building out my personal network. However, I generally believe that creating content is a time-inefficient way to build your network, which is discussed in Chapter 12. If you're excited to write and speak, then absolutely do that; it will help you build your network. But if you just want to build connections, then I strongly recommend pursuing other avenues.

In the time it takes to write a single decent blog post, you could send 20 cold outreaches and attend three community events. If you try to make writing more efficient by writing shorter pieces or editing less, that probably won't work too well either. Content-driven networking is more about writing interesting or thought-provoking pieces (which takes considerable time) than it is about writing at a high volume.

LARGE COMMUNITIES

There are a number of very large engineering leadership communities out there, most notably the Rands Leadership Slack. These are good places to learn, and they can be a good place to meet executives to add to your network, but my experience is that they struggle in a couple ways from an executive's point of view (which is in no way to their discredit, this is not their goal):

They are too open for a genuine discussion.

It would be irresponsible to have a discussion about your circumstances in a forum where members of your team could easily view that discussion. When I have particularly good discussions about an executive problem, it always involves a significant amount of detail that I absolutely cannot share in a public forum. There's no viable way to have these conversations properly outside of a private space.

There is an inverse relationship between ease of access and quality of advice.

There's no provable law here, but it's generally my experience that the people with the best advice and relevant experience are too busy doing things to be consistently hanging out in large communities. There are exceptions here—and if you hang out frequently in a public community, then I'm sure you're one of them—but I am deeply skeptical that you get the best advice on complex issues by asking questions in large forums.

There are always details in the margins on a subject this broad. When does a small community cross into being a large community? What if the community is heavily moderated and has confidentiality in its code of conduct? Sure, there are absolutely some communities that make these things work, but I remain surprised by the degree to which things leak even from nominally private spaces. This is even more true for you as an executive, which makes you an interesting topic of gossip.

WHAT DOESN'T WORK

If you're finding that these approaches consistently don't work for you, then get someone to review how you're engaging. It's normal for any given request to get dropped—people are busy—but it's quite unusual to have requests consistently ignored if you follow the approaches discussed in this chapter. The best bet is that you're doing something wrong or have already done enough wrong things that your network isn't motivated to help you.

Three patterns that are particularly unlikely to work well for you are:

- Ambiguous requests, particularly for time ("could we grab some time to chat?"), make it hard for the other party to help, even if they are motivated. Be specific and concise.
- Confusing requests that force the recipient to decode the message. If you're asking for help, you should spend the time making a clear request

rather than forcing the other person to interpret and guess at your meaning—you're already asking them for something, don't ask for two things.

- Asking for introductions or connections when there is not clear, real mutual value. Otherwise, you're burning the recipient's capital without a return. It also makes for an overall awkward interaction, so the recipient is less likely to help you in the future.

While I think of these as obviously bad patterns, I get these requests surprisingly frequently, and they're hard to answer even when I want to help.

Other Kinds of Networks

In addition to building your network of engineering leaders, there are a few other networks worth investing in as well: founders, venture capitalists, and executive recruiters. Each group will be useful in a different aspect of your work and require a different approach to build the relationship.

FOUNDERS

Founders are a powerful, oddly shaped resource. In their business area, they often have unique, deep insight into the future. They are often widely connected with other founders. They will rarely know more about engineering or management than you do, but they often bring a unique perspective that is similar to the founders at your company. If you're struggling with your own founders, other founders are particularly helpful to chat with.

The best way to meet founders is to work in fast-growing, entrepreneurial companies. The second best way is angel investing. A small amount of angel investing goes a surprisingly long way, as founders tend to be well connected with other founders.

VENTURE CAPITALISTS

Venture capitalists are the most connected people and best pattern-matchers in the industry. While they can't necessarily evaluate engineering leaders, they can introduce you to 10 engineering leaders within their portfolio. While they can't tell you the right compliance automation tool for your company, they can tell you what their portfolio companies use, and in particular what their best companies picked. As long as you're asking a question that benefits from industry connections or pattern awareness, there is no one more helpful than a venture capitalist.

One of the many perks of working at larger and well-known technology companies in a high-tech center like Silicon Valley, New York, or London, is that

a decent number of your colleagues will drift into venture capital over time. This won't happen overnight, but venture capital firms frequently hire experienced industry operators, and you'll find that you know quite a few folks over the course of a decade.

If the slow approach isn't working for you, I particularly recommend putting your CEO to work connecting you with a few venture capitalists they know, likely on your board, which will help you better understand their perspectives on your company, in addition to growing your network.

EXECUTIVE RECRUITERS

Executive recruiters are an integral part of getting a job as an Engineering executive (as discussed in Chapter 1), and also tend to have a good understanding of talent migration across the industry. You'll build an executive recruiter network over time from working on hiring loops. If you don't know any today, reach out to your network and you can almost always get a connection. Introductions to recruiters tend to be less fraught than other introductions because recruiters typically actually want to be introduced.

Summary

In this chapter, you've spent time understanding why your network is a collection of relationships, and that relationships require intentional cultivation. To develop your network, set a small goal, like meeting one new person each month, and slowly build your network up over time with a focus on the challenges you're working on at that moment. Don't make it your top priority, but don't forget it either.

Find links to further reading and resources on *https://lethain.com/eeprimer-refs-15*.

Onboarding Peer Executives

While many companies build out an elaborate Engineering onboarding program, the process for onboarding new executives tends to be an ad hoc, chaotic affair because it's used too infrequently to ever become excellent. Part of the problem is similar to that of an executive job search: every executive and role is unique, and it's hard to create a repeatable program to handle one-of-a-kind onboardings.

The other part of the problem is that the executive's manager, the CEO, usually hires a new executive to solve a specific problem. Once the executive starts, the CEO will often turn to focus on their next biggest problem. This happens so frequently that the two most common frustrations I hear from executives about onboarding their peers are, "Isn't this the CEO's job? Why should I worry about it?" followed shortly thereafter by, "Why didn't I worry about it more? I knew leaving it to the CEO was a mistake."

Even when the CEO is engaged, most new executives want to impress their CEO. You'll often find new executives struggling a bit with onboarding, while simultaneously reassuring the CEO that they're doing fine. Your job as a peer executive is to help the new executive succeed, even if no one tells you it's your job.

In this chapter, I'll break down onboarding peer executives into five topics:

- Why executive onboarding matters
- How onboarding executives differs from onboarding engineers
- How to share your mental framework
- Defining your roles in respect to one another
- Investing in time to work together on an ongoing basis

Peer onboarding, like a surprising number of good executive practices, is straightforward and easy to do well with some time and attention. By the end of this chapter, you'll have a clear structure for onboarding a new peer executive.

Why This Matters

It's unfortunately true that you can be *perceived* as a high-achieving executive without accomplishing much. However, you cannot genuinely *be* a high-performing executive unless you, your peers, and your CEO are all doing excellent work that culminates in a successful business over an extended period of time. You can create whatever narrative you want about your extraordinary Engineering staff whose excellence was masked by the weak Product or Sales teams, but you'll only be exceptional if every part of the executive team functions well.

Your best tools for forming an effective executive team are working with a team-oriented CEO and being deliberate in the executives you hire. Once you've hired someone, your best remaining tool is effectively onboarding them. Without a good onboarding process, even strong executives will make missteps, sometimes unrecoverable ones. These issues are much easier to prevent than to recover from.

Onboarding Executives Versus Onboarding Engineers

As an engineer, you'll often be responsible for onboarding peers onto your team. In those cases, you're often implicitly more senior than the peer you're onboarding, but you're both knowledgeable in the shared field of software engineering. Onboarding executive peers is different. You're truly peers, without any implicit hierarchy between functions, and you often don't share the context of having worked in a common profession because they might be the new executive for Finance, Sales, or any other function outside Engineering. Further, engineers (and most other functional contributors) go deep and narrow, whereas executives need to start by going shallow and broad.

To tease out the distinction between onboarding engineers and executives, consider these two points:

When onboarding a peer software engineer, your goal is to help them understand your current process and one specific onboarding project well enough to implement that one project.

After two or three such projects, the new engineer will be relatively ramped up. The best indicator in Engineering onboarding is for the new hire to complete initial projects quickly at a high-quality bar, which is typically most dependent on their ability to navigate a large, lightly documented codebase for the first time. Bad signs are making very slow progress on projects, as well as arguing about existing processes before ramping up on them.

When onboarding a peer executive, your goal is to help them understand the company's landscape (business, team, project, and process), with a particular focus on the most critical, current issues.

Because there is so much to absorb, they should get a very broad overview and then go deep as required by the immediate fires they need to put out. Good executives will immediately focus on the highest-priority areas, check their plan with nearby peers before beginning implementation, and bring a positive energy (even if things are an absolute mess). Bad signs are inaction on key issues, changing areas that are already working well before addressing higher priorities, or immediately falling back onto previous companies' approaches that fit poorly in their new circumstances.

Sharing Your Mental Framework

Calm started by providing an app for meditation and we had many users write in to tell us how that app changed their lives for the better. Five years later, we were a much more complex business, but many folks on the team still operated as if we were that original meditation app company. This wasn't because they were unaware of the expanded business. Rather, their mental framework held on to the earlier iteration of the company.

Once you apply a given framework to a company, shifting it is quite challenging; this makes helping new executives build an accurate mental framework of the business particularly valuable. It also means it's something you have to do in that executive's first few weeks on the job.

If you're onboarding a new peer executive, I recommend covering all of these topics in the first two weeks, even if it means canceling some of your standing meetings to make it possible:

Where can the new executive find real data to inform themselves, rather than relying on existing narratives?

The best executives will listen to you but won't fully believe anything until they're able to find data to substantiate your perspective. That's not because they don't trust you, but because any seasoned executive has been burned by trusting someone who fervently believed something that ultimately wasn't true.

Examples: Company dashboards. Board reports. Tableau. Looker. Google Analytics.

What are the top two to three problems that they should immediately spend time fixing?

This is traditionally the first question to answer, but I think setting the mental frame is even more important!

Examples: The CEO is very upset with the pace of execution. Friction among Product, Engineering, and Legal is preventing forward progress on new initiatives. Personal conflict between two senior leaders has frayed cross-functional relationships.

What is your advice to them regarding additional budget and headcount requests?

The classic new executive move is to privately request more budget from the CEO, who agrees to keep the new executive happy, which often creates a cascade of cross-functional hiring that can significantly impact the company's cash flow. If you don't want them to immediately request additions to the headcount, you should have an open discussion with them about the consequences.

Examples: Although I suspect the CEO will approve more headcount if you ask today, if you look at Product hiring relative to revenue, we've gotten significantly less effective each year over the last three years. I recommend spending time debugging why we're getting less efficient before further increasing the size of the Product organization.

What are areas in which many companies struggle, but that are currently going well here?

Specifically, you want to steer the new executive away from running their "default playbook" where things are already working.

Examples: Product and Engineering are already planning together effectively. The CEO is very pleased with the release cadence (perhaps they are instead frustrated that the cadence isn't translating into revenue).

The collaborative headcount process is already reaching reasonable cross-functional equilibrium.

What is your honest but optimistic read on their new team?

Give them a detailed, transparent look into the members of their team while keeping in mind that any historical gaps in performance may have been connected to the reason the new executive was hired.

Examples: Two years ago, Design was an active part of planning product work. Then the head of Design moved from an embedded model to a centralized studio, and the designers generally lost context on user and business goals. In particular, the impact I've seen on the Design leads has been quite high, and some of the highest-impact folks I've worked with are now feeling a bit disengaged.

Who do they need to spend time with to understand the current state of the company and its implicit power structure?

Be sure to introduce the new executive to the longest-tenured employees, to those who uniquely hold parts of the business in their heads, and to individuals who have significant influence over the executive team that wouldn't be obvious from the reporting hierarchy.

Examples: The Head of Quality Assurance or Customer Experience. A long-tenured Product manager. The third founder who isn't in a managerial role but who remains very influential.

What is going to surprise them?

In particular, discuss things that are done in a unique way at the company, but also talk through quarterly or annual processes that have schedules they might be unaware of.

Examples: Business reviews. Quarterly or annual planning. Written documents for every meeting.

What are the key company processes?

How long have they been in place, and how are they working? How should the new executive propose changes to any of these processes if they feel they're necessary?

Examples: Planning. Budget and headcount. Performance management. Cultural surveys.

There are, without a doubt, peer executives who will only listen to guidance from the CEO. I've been responsible for onboarding multiple executives who fully ignored my advice and immediately advocated for a full rewrite of our product or technology stack. I felt pretty bad about not influencing them more successfully, but nonetheless glad that I'd tried my best to do so. Although it's hard to quantify, I'm confident I've prevented many nightmares by proactively sharing my mental frameworks with executives before they leap into action.

Partnering with an Executive Assistant

One of the magical moments you'll experience as a senior leader is the first time you get support from an executive assistant. It's also a confusing moment for many executives, who find themselves figuring out how to onboard a new assistant, and even when to consider hiring one. There's a real learning curve to partnering effectively with executive assistants. Here are some suggestions on hiring an assistant and working together with one:

When to hire

While CEOs often hire an executive assistant before the company reaches 30 folks, it's uncommon for other leaders to get an assistant until their organizations reach 50 or more people, and often assistants are shared until groups reach 200 or more. The details will vary a bit by company, but executives should aim to have an assistant partnering with their organization by the time it reaches 100 people. This role goes beyond supporting you to being a partial resource to the leaders on your team who are also managing large teams.

Leveraging support

When you first start working with an executive assistant, it can be hard to figure out what to offload to them. It can also feel weird, since these are often tasks you've personally deprioritized—probably because you've felt they're less important than the work you're doing—and now you're asking someone to do them for you.

Tasks that typically delegate well are:

Managing time

This involves managing your calendar, reviewing time allocation and suggesting improvements (maybe that weekly meeting really isn't that useful anymore), and also routing requests to maximize focus and minimize interruptions.

Drafting communications

Coordinating clean communication, especially for sensitive organizational changes, requires great attention to detail and making sure you loop in the right folks.

Coordinating recurring meetings

At some point, most companies end up with recurring operational meetings and executive assistants can coordinate the agenda, presenters, timing, and action items.

Planning off-site sessions

A great team offsite will gel a team and provide focus in your execution, and an executive assistant can do a great job of alchemizing a general theme into an agenda, a rented room, and a date.

Coordinating all-hands meetings

Most organizations bring the company together to talk about their work, progress, and priorities, and an executive assistant can coordinate the presenters, give feedback on the presentations, run practice sessions, and coordinate execution of the meeting itself.

That said, the partnership between a leader and an executive assistant is more custom-fitted than most, and ultimately you should find the intersection between what you need, what works for both of you, and the executive assistant's career development.

Being supportive

Executive assistants do a tremendous amount to make leaders successful, but the social contract goes both ways. If you want a tremendously impactful executive assistant, then approach your relationship as a partnership built on trust and communication. Folks who don't support their EAs at best get a pair of hands instead of both head and hands, and at worst find themselves rebuilding their partnership with a new EA every couple quarters. While the EA's role is to support you, they can't do that unless you support them as well.

Define Your Roles

Effective Engineering organizations start with strong relationships across Product, Engineering, and Design, and those relationships are greatly influenced by how well their executives work together. Many executives start their new role assuming that it's more or less their old role but in a new company. This leads to implicit assumptions about who does what that may be at odds with how the company currently works, which results in conflict around role definition. You can't prevent all tension between you and other executives about your respective roles, but you can avoid accidental friction by discussing the areas of intentional disagreement.

The three questions you should work together on are:

What are your respective roles?

Both of you should independently write down your expectations of the other role, then share those expectations with one another. If you find something that you disagree on, that's great! Either it's an accidental disagreement that is quick to resolve, or it's a genuine disagreement that the two of you will need to spend time working through. In either case, you get to skip the step where you're surprised to learn about it.

How do you handle public conflict?

Part of being an effective executive team is appearing aligned with each other, even when you're not, and that depends on having a clear communication strategy. This is easiest when you agree on how you handle public conflict, usually by trying to politely resolve it in the current meeting if it's

easy to do so, or agreeing to take it to a private meeting if you're far enough apart that you can't resolve it without exposing the conflict to the wider team.

What is the escalation process for those times when you disagree?

It's inevitable that you will disagree with each other, sometimes on very important topics. I have certainly disagreed with my executive peers on budget allocation, the roadmap, acquisitions, and many other topics. Most executives stumble through agreeing on an implicit escalation process for resolving conflicts between each other, but you can also just have a direct conversation about it. When people complain about politics, often it's a subtle disagreement about an implicit escalation contract: you can just make it explicit instead.

These jobs never stop changing. The relationship between you and your peer will be the only constant in navigating those changes. Spending time up front to clarify roles and establish open communication will take a bit of time but will save a tremendous amount of time longer term and will start your relationship off on the right foot.

Trust Comes with Time

Once you've agreed on an initial definition of your roles and how you'll work together, the final step of onboarding is to schedule ongoing time working together. Your initial agreement won't actually be quite right, and you'll only figure out the issues by working through difficult scenarios together. Executive calendars fill up so quickly that, unless you book recurring time in advance, you'll only work together on very painful issues that escalate past your routine calendar, which are the sort of topics that strain rather than build an early relationship.

To establish trust, the key practices I'd recommend are:

Spend some time knowing them as a person.

Schedule some lunches or dinners to create a few non-transactional experiences together with your new coworker. Executive roles often pull you toward treating each other transactionally, but it doesn't have to be that way. Try to make this happen a few times a year.

Hold a weekly hour-long 1:1, structured around a shared 1:1 document.

Repeated exposure has a magic to it, and an hour is enough time to dig deep into two or three meaningful problems. If a weekly hour feels like too much, it may be a sign that either your functions don't work together much or that your company is at a stage where executives are leading in the weeds, which many early-stage companies are. Try the weekly hour a few times, but absolutely move to a less frequent cadence if this doesn't feel productive.

Identify the meetings where your organizations partner together to resolve prioritization and work through any conflict.

These might be weekly metrics reviews, monthly operational reviews, quarterly planning meetings, or something else. Once you identify these meetings, make sure both of you attend at least a few sessions together and afterward discuss any conflict or disagreement to be better aligned in the future.

Finally, trust is about doing what you say you will, which takes time to build. You can absolutely immediately enjoy working with someone, but you can only trust them after having them make commitments and seeing them live up to those commitments. This can't be rushed.

How Much Progress Is Possible?

If an engineer is unconstructively argumentative or does particularly poorly in their first three months, you would generally manage them out of the company. Struggling executive hires are often evident within the first three months but take a year or longer to leave a company. Mediocre but very nice executives last much longer, often remaining indefinitely.

As a peer, your only bet is to partner in good faith, trying to help new executives succeed. Teach them the mental frameworks you use to navigate your company. Work with them to define adjoining responsibilities and how the two of you will collaborate. Make time to invest in the relationship, even if it's a bit rocky or doesn't feel like it directly addresses one of your current priorities. These are small steps, but together they'll help the new executive build their foundation at your company.

Even the best executives need support and you can significantly change your company's trajectory—and your experience working within it—by helping them onboard effectively. Even if a new executive isn't working out, it's the CEO's job

to address that, not yours, and pushing too hard will only reflect poorly on you without solving the problem. We'll discuss this further in Chapter 18.

Summary

In this chapter, we talked about onboarding peer executives and why onboarding your peers is an important mechanism for making you—and your executive team—more effective. We also dug into how to tune your approach to onboarding executives and why it's quite different from onboarding engineers. Finally, we spent time on the three most important areas to prioritize: sharing your mental models, being explicit about how your roles intersect, and making an ongoing investment in your new relationship. New relationships are defined by your first interactions but they're sustained by ongoing attention: doing a bit of both goes a remarkably long way.

Find links to further reading and resources on *https://lethain.com/eeprimer-refs-16*.

Inspected Trust

The path to becoming an Engineering executive used to start with the transition from writing software as a software engineer to serving in a hybrid software engineer/people management role. That role was typically called tech lead manager, and combined being the team's senior-most technical contributor along with fulfilling the team's people management duties. Many folks struggled with that transition and fell back onto the familiar habit of technical leadership to avoid the discomfort of developing the new skills necessary to succeed in people management.

Since then, the industry has attempted to solve that challenge by decoupling technical and people leadership into distinct roles. Today, it's common for first-time managers to get the advice that they should immediately stop writing code at work. Writing code, the argument goes, is a distraction from the more important work of learning to manage a team. In this world, managers should step back from their team's work and focus on being people managers. This advice is summarized into one short phrase that most managers get early in their management career: "Trust your team."

This advice is almost correct, which is why it's so easy to misapply. What managers—and executives—truly need isn't unquestioning trust but inspected trust, which is a trust-but-verify approach. In this chapter, we'll discuss:

- How executives may rely too heavily on trust, and how that undermines their leadership
- Why trust isn't a management technique
- Some tools to use instead of relying exclusively on trust
- Some recommendations for introducing inspection into low-inspection organizations and companies

Leading effective and collaborative teams requires a nuanced approach to trust. As an executive, you absolutely should work to establish trust with your team and organization, and this chapter will help you build trust by avoiding the temptation of relying on trust alone. You'll particularly benefit from this chapter if you view trust as one of your fundamental leadership techniques, which is a well-meaning approach that I've seen undermine many talented leaders.

Limitations of Managing Through Trust

When you join a company as an executive, you join with a large reservoir of trust from others based on your having successfully navigated the hiring process. Why would the company hire you if you weren't worthy of their trust? You'll be most effective if you start by understanding your new environment (as discussed in Chapter 2), but a surprising number of new leaders jump into their new environment somewhat recklessly. That initial stockpile of trust doesn't go particularly far if you start burning it immediately. You may expect your new manager will step in to set expectations or course correct your approach before you damage your credibility, but it's surprisingly common for new leaders to cause a great deal of damage before their manager provides corrective feedback.

There are many reasons why your well-intentioned manager might not give you the feedback you need, and the most frequent reason I see is that your manager is trying to manage through trust. Their argument is that it's empowering to give you room to make mistakes while ramping up in your new role; to do otherwise would be stifling you when you need support the most.

They'll keep talking about empowerment as you thrash around your new organization, lose trust with your peers, and eventually dig yourself into a hole that's too deep to dig out of. A year later, you'll be a known quantity, in particular a quantity who is known for having rather poor execution. At that point, you'll probably drift out of the company with some quiet angst against your manager. Your manager will mark you down as unregretted attrition, sigh about how hard it is to detect low performers when interviewing, and restart the "manage-through-trust" antipattern with a new candidate.

This scenario should be a hyperbolic example, but it took me about five minutes to think of 10 cases of this happening in my personal experience. Some of them, I'll admit, were scenarios where I was the trust-embracing manager who inadvertently undermined the new hire I intended to support. This is the fundamental risk of relying too heavily on trust. You will feel like a good manager

along the way, but you're avoiding accountability and undermining your team while convincing yourself that you're providing avenues for growth.

Trust Alone Isn't a Management Technique

While you've likely been trained to believe that trust is a core tenet of effective management, trust on its own isn't much of a management technique. Trust cannot distinguish between the good and bad variants of these four classifications of your direct reports' work:

Good errors
: Good process, good decision, bad outcome (e.g., missing a project's release date because a vendor had an outage that day)

Bad errors
: Bad process, bad decision, bad outcome (e.g., missing a project's release date because you shifted unfamiliar engineers onto the project and predictably slowed it down)

Good successes
: Good process, good decision, good outcome (e.g., hitting a project's release date through good planning and execution)

Bad successes
: Bad process, bad decision, good outcome (e.g., hitting a project's release date because a vendor unexpectedly released functionality that the project was missing)

You'll have a very difficult time identifying and growing your management bench if you can't evaluate judgment separately from outcomes. Without a clear ability to distinguish between those classifications, randomness will be a stronger influence on who you consider to be an effective leader than their actual effectiveness.

Why Inspected Trust Is Better

Don't get me wrong; trust is a necessary and powerful component of effective management. It's blind trust and a lack of intervention that I'm warning against. What I recommend instead is *inspected trust*, which is a "trust first, then investigate before acting" approach.

Some examples of inspected trust include:

- If a member of your team mentions that another team is causing problems, trust that there is a problem. Then, find a way to verify the problem independently and explore alternative explanations to consider. How do other teams involved feel? Is there missing context from a third party that might be preventing the other team from engaging? What are options for addressing the situation?
- If a manager mentions someone on their team is causing problems, trust that there is a problem. Then, once again, find a way to independently verify the problem. Have a skip-level meeting with the individual. Have lunch with that person and their team. Review the latest reports for the team's projects. Form your own opinion on potential solutions.

Once you've independently validated the situation and considered options, you can then engage with the individual who raised the concern and confidently solve the problem together. Their read on the scenario might have been accurate from the beginning, but they may have also been missing important context, or even been a significant part of the problem. This inspection can be quick and it allows you to swiftly move to problem solving without lingering uncertainty.

Occasionally, I've had folks push back on me inspecting their work, arguing that I should just trust them.

I understand the short-term perspective, which is that it feels good to be trusted without question, but long-term it's much more important to ensure you're properly solving a real problem than leaping to a solution without validation. Earlier in my career, I felt bad when folks viewed my inspection as a lack of trust, but now I view that concern as an indicator of a deeper relationship issue between me and the individual I am working with.

Your goal in working together is to help each other succeed, and that includes catching each other's mistakes. Inspection is a gift: It's taking the problem seriously enough to ensure we're getting to the bottom of the actual issue at hand.

Note

Providing inspected trust isn't strictly something that managers do for their teams; many of the best folks I've gotten to manage have diligently inspected everything I told them, refining my rough proposals into something much better and catching me when I've gotten too attached to my own interpretations.

I've also found that some of the leaders I've managed push back on the idea that they should inspect their team's work. Usually, they were concerned that inspection signals a lack of trust in their team. I made the same argument to them that I am making to you here: reliance on uninspected trust can feel very comfortable, but it's ultimately an approach that undermines your team leaders and the folks they manage. Even the best folks you ever work with will occasionally be wrong despite following a good process and making good decisions. Uninspected trust misinterprets good errors as being untrustworthy, and it's hard to build long-term relationships when you routinely mistake good actions as untrustworthy behavior.

Inspection Tools

If I've convinced you to move toward inspected trust rather than uninspected trust, then the next question is to figure out an appropriate set of inspection tools to support that approach. There are quite a few effective styles of inspection, and experienced executives should be particularly comfortable with one or two of these tools while also being capable of using the full toolkit. The inspection tools that work best will depend a bit on personal leadership style, the particular company, and the specific problem at hand.

Some inspection tools that I've seen used to good effect include:

Inspection forums
: The default inspection strategy for established companies is setting and tracking goals, typically in a weekly or monthly metric review forum. Quarterly review forums are too slow to serve as a principle inspection tool (things can go very wrong in a quarter, especially in a new area or with a new leader).

Learning spikes
: After getting a sense of a problem, drill deeply into the problem by talking to as many folks involved as possible. Focus on talking to folks across functions and to whoever is directly doing the work. Avoid spiking with folks whose job it is to summarize other people's work such as senior managers, although they are often the best source for a list of others to talk with. Learning spikes work because they bypass the existing information hierarchy. Hierarchy is great at condensing too much information into a narrative, but narratives are crafted with select information. Selection

implies omission, and sometimes a well-meaning narrative omits essential data. Avoid all this by going straight to the source.

Engaging directly with data
: Many senior leaders rely exclusively on heavily prepared data to understand challenges, but prepared data includes the bias of its preparers. Instead, accumulate a small set of data sources whose preparation you understand in detail (what is omitted, what is included, and what kinds of bias that preparation potentially introduces), and cross-reference those sources against new information presented to you.

Having a fundamental intolerance for misalignment
: Some leaders have a deep fundamental intolerance for confusion and misalignment. They'll hear a slight bit of confusion in a meeting and keep digging on it until they understand what caused it. This ongoing intolerance surfaces and resolves the sorts of problems that—if left unchecked—would undermine trust.

If you're working in, or aspiring toward, a senior leadership role, I'd encourage you to actively practice all four of these inspection approaches. After you're comfortable with all of them, it's fine to lean more heavily on those that feel more natural as long as they fit with the problem at hand.

Incorporating Inspection in Your Organization

If you're joining a new company where inspection isn't the norm, or if inspection is a new approach that you're introducing after previously leaning toward uninspected trust, I have a few suggestions for incorporating this strategy:

Don't expand everywhere all at once; instead, focus on one or two critical problems.
: You will slow everyone down and dilute your focus if you go too wide. Even if the ideas in this chapter feel obvious to you, remember that there's a learning curve for you and the team around you.

  Inspection wasn't the norm when I joined Calm, so I focused on two particular problems. First, teams were having trouble planning work because every feature required aligning schedules across iOS, Android, and frontend and backend engineering teams. Digging deeply into that issue culminated in a new project-oriented team structure rather than the expertise-oriented team structure. Second, a handful of senior engineers were in semi-open conflict about our technical direction. I drilled into

those conflicts, which resulted in me documenting several pillars of our technology strategy to force alignment.

Figure out your peer executives' tolerance for inspection forums before trying to roll them out.

All companies introduce deeper inspection forums over time—perhaps a business review, a product review, a pre-meeting before the board meeting—but these meetings only succeed with buy-in from your CEO and executive peers. Without executive engagement, these meetings require a great deal of preparation but won't create outcomes, which is a waste of everyone's time.

At Uber, we had an ex-Google executive join Engineering who mandated OKRs and OKR reviews within our area of the team. This is a valuable technique, and we certainly needed more inspection than we had at the time, but because the executive peers had little interest in participating in the process, it was a limited success.

Explain to your Engineering leadership team what you're doing and your thinking behind it.

Being explicit about your rationale will go a long way, so tell them that your inspection is a long-term investment in their success, not a demonstration that you don't trust their work.

At Stripe, we rolled out our inspection norms via business review meetings, but didn't frame the goal of inspection as creating trust. Without that explicit framing, they were often viewed as procedural meetings that drove micromanagement, rather than recognized as enabling meetings that empowered teams to move quickly and avoid long-term misalignment.

Even if you roll out new inspection norms exceptionally well, you should be prepared to run into some defensive reactions from team members who view inspection as a lack of trust. It will be tempting to change your behavior based on that frustration, but resist the temptation. In the long run, the best members of your team want to do good work to solve real problems and they want to get feedback to continue improving. If there are members of your team who are so uncomfortable about making errors that they can't receive feedback, then they aren't the right leaders for your organization.

Summary

Don't come away from this deciding that you shouldn't trust your team. You absolutely *must* trust your leadership team to scale as an executive. Even more importantly, though, I hope you appreciate that merely trusting your team is abdicating from one of the most important parts of your role: helping your team succeed. Use the techniques here to shift away from relying exclusively on unvalidated trust, and instead build more robust trust with your team by pairing trust with inspection to detect errors early and learn from those errors. Inspected trust is work and it takes time, but it culminates in an organization that polishes leaders for the better. The alternative appears to work in the short run but leaves you with a rotating cast of short-term leaders and a low-trust environment.

Find links to further reading and resources on *https://lethain.com/eeprimer-refs-17*.

Calibrating Your Standards

Managing teams has taught me a lot about my own behaviors and motivations. For example, I overworked for a long time. This left me continually teetering on the brink of burnout and I had no energy left to absorb the sorts of organizational changes that happen at any company. Despite doing good work, I handled change poorly, and I picked up the reputation for being difficult to manage.

I'd like to say that I learned from my mistakes directly, but a more honest version is that I came to understand this dynamic mostly through working with folks struggling with the same issue. After seeing it in others several times, I came to notice it in myself. This culminated in joining Stripe with the explicit goal of pacing myself to be more valuable after four years, rather than getting exhausted after two years like I had at Uber.

In this chapter, we'll dig into:

- How high standards, generally considered an obviously good thing, frequently cause friction for executives
- Why it's important to match your standards with the standards that your organization upholds, rather than assuming that high standards are obviously right
- How escalating in regard to perceived low standards can get you in conflict with your CEO
- Why role modeling is an effective tool for shifting an organization's standards
- What to do when your standards are not serving you well in your current situation

By the end, you will start to recognize when your colleagues' standards exceed your company's tolerance for standards (perhaps you'll even notice this about yourself), understand why it's important to tune your standards to your environment, and have several options for trying to address issues where you believe your organization's standards are miscalibrated.

The Peril of Misaligned Standards

Overworking is an interesting vice because it's socially acceptable and some view it as a necessary precondition to outsized success. The category of "socially acceptable professional vices" is an interesting one because these vices will hamper your career progress in non-obvious ways. The socially acceptable vice that I've seen harm the most executive careers is having higher standards for those around them than their organization supports.

It's a truism that you always want to hire folks with very high standards, but organizations only tolerate a certain degree of those expectations. If you go further, rather than upholding you as a bar raiser, your organization will view you as the problem. I've seen this happen *so many times* in my own work, in friends' work, and in emails that wind up in my inbox.

A few examples:

- An interim vice president of Engineering (VPE) is at a company whose CEO won't finalize the VPE role because a peer is upset they didn't get it. That peer has been struggling for some time, but the CEO doesn't want to "rock the boat" so leaves them both lingering. The interim VPE's attempts to hold their peer accountable are viewed as "evidence they're not ready" for a permanent VPE role.
- An Engineering manager works with a Product manager whose proposals are both very expensive to implement and misaligned with the company's goals. The Engineering manager flags the issue to Product leadership and it gets reframed from a concern about the Product manager's performance into an issue of two peers not collaborating well. Both are pushed to "collaborate better" but the team's impact remains poor.
- Engineering directors work at a company that has instituted company-wide bar raisers because one of their peers was unwilling to maintain the shared hiring bar. The CTO was unwilling to hold that director accountable, so

the other directors followed the only solution they could think of that wouldn't be interpreted as "interpersonal conflict" by the avoidant CTO.

As you look at enough of these scenarios, you can identify a common pattern. The main character is trying to do their job effectively, but can't because of the low performance of a peer. They escalate to the appropriate manager to address the issue, but that manager decides to view the performance issue instead as a relationship issue: it's not that the peer isn't performing, it's just that the two of you don't like each other. Instead of being the manager's responsibility to resolve the performance issue, it's now the main character's responsibility to improve their relationship. By attempting to drive accountability in their peer, the main character has blocked their own progress ("you're just hard to work with") without accomplishing anything.

Matching Your Organization's Standards

What I've come to appreciate is that the underperformer's manager is almost always aware of the underlying issue, and for some reason they're simply unwilling to confront it. You might think that you're bringing a new problem to the CEO to solve regarding another executive, but what you're actually doing is trying to hold the CEO accountable for not solving a known problem, which they may not appreciate.

Further, even generally excellent companies have areas that operate with lower standards. Sometimes this is accidental, but it's often deliberate (e.g., a small Quality Assurance team that's only intended to do mobile QA for one legacy application). In that scenario, you believe you're holding the CEO accountable for a mistake, but from another perspective they simply believe the given problem merits a different quality standard than you do.

I worked at a company where the Growth Engineering organization was viewed as a second-tier organization that had a lower hiring standard and executed to a lower quality bar. On the other hand, the team moved extremely quickly, tolerated frequent changes in direction with good humor, and generally solved problems that the wider Engineering organization didn't want to work on.

At the time, I felt this was a sign of poor leadership, but I've evolved my perspective a bit. This didn't make for a particularly harmonious working environment, but it did effectively achieve the company's goals. You and your company have a finite amount of capacity to allocate, and sometimes hard deadlines to meet. This combination sometimes means doing some work at a lower standard.

Escalate Cautiously

If you're struggling with your peers not meeting your standards, it's important to recognize that you're disagreeing with how your CEO is doing their job. That's a normal thing to happen, but it requires a careful touch. If you still want to have a direct discussion, you can usually have a conversation about why they're comfortable maintaining the status quo. You can even add more context about how it's impacting you or the team you work with. Sometimes this will lead to change, but it's a slow process. You're usually not going to change their mind overnight.

What I've found effective in these cases is to lead with constructive energy directed toward a positive outcome. Even if you can't get your peer executives' performance addressed directly, you can often overcome your peer's bad performance by generating excitement in the direction you want to go. Enough excitement will give the resolution-avoidant manager a way to solve their problem without actually engaging with the situation that they're unwilling to address.

I know that folks don't want this advice, but I'm going to put it out there: it *is* unfair to be forced to overcome a peer's poor performance—that's the manager's job—but work isn't a perfect place. Success is finding a path forward among the options that actually exist. Grinding yourself down in frustration about the paths that don't exist doesn't solve anything. It may take a few tries to learn this lesson. I certainly devoted quite a few years to learning it the hard way.

Of course, there are situations in which you may feel ethically obligated to raise a concern, and you certainly should in those cases. I just caution you to recognize when you're taking action out of ethical obligation as opposed to when you're taking action to improve your working situation, and to push you to avoid framing a disagreement about standards as a disagreement about ethics.

Role Modeling for Your Peers

If there's a particular process or area that you care about, one that your CEO is open to improving but unwilling to invest in, the most effective path I've found is visibly role modeling good behavior. I did this frequently enough as a middle manager that I ended up developing a playbook that I call "Model, Document, Share".

The core playbook is:

Model
: Identify the area you want to improve and go about role modeling a high-standards approach. This might be drilling into a problem area, running a hiring process that considers internal candidates, or running manager training for your team. Iterate a bit until you're confident your approach is working well.

Document
: After you've gotten the approach working, write it up into a clear document. Write up how to follow your approach, but also write up the rationale behind it, ideally from the perspective of evaluating whether the approach works rather than one of certainty that it's a better approach.

Share
: Send the documented approach to the teams you want to influence, letting them know what you've tried, what you learned, and why it was an effective approach for your team. Engage with those who show interest, and let it go for teams that don't engage.

This approach relies on exciting teams and leaders who agree with you about the problem, and who are interested in improving on the status quo. Without authority, which you explicitly don't have in this case, you won't make much progress with teams that actively disagree, which is fine. It's a useful technique for trying to make progress in areas where authority figures have decided not to make improvements, without alienating those authority figures. Keep in mind that you won't have the tools to solve every conflict.

Adapting Your Standards

Early in my career, I developed the idea of "lessons not worth learning." One of those lessons I placed in that category was lowering my standards. My rigidity around high standards, I believed at the time, was going to make me successful. As I've gotten deeper into my career, I look back at that early phase with some nostalgia, but I also view my perspective as a bit hubristic. Steering a long career requires more flexibility and adaptability than I ever imagined necessary as a 25-year-old. Sometimes work is your priority, but other times your personal life is a chaotic mess and you'll spend much of your energy supporting your family or friends.

When you run into one of these circumstances where your standards are higher than the organization's—or potentially just your CEO's—spend time figuring out what's most important to you at that moment. The temptation to make noise may never go away, but the consequences of such behavior will only grow as you operate in more senior roles. While I still believe that some lessons aren't worth learning, I've also come to appreciate that some penalties aren't worth paying.

Summary

In this chapter, we explored the paradox of high standards being both desirable and a frequent source of conflict for executives with their peers and CEO. Your standards must be aligned with your organization's tolerances. You're now equipped to think through where high standards are serving you and where they may be undermining your relationships. Don't abandon your standards, but also make sure they're supporting you toward your goals.

Find links to further reading and resources on *https://lethain.com/eeprimer-refs-18*.

How to Run Engineering Processes

Uber ran a tech spec review process called the DUCK Review. Although "DUCK" didn't stand for anything—it was created as a deliberate non-acronym—this was otherwise a fairly typical review process. When I first joined, we'd review one or two specs each week. The volume of requested reviews kept growing, and six months later there was a one- to two-week delay between requesting a review and receiving feedback. A year after that, the process was disbanded due to lack of bandwidth to process all the incoming specifications. This was an instructive experience for me, because the DUCK Review produced very valuable feedback but ultimately still failed because its operational costs were higher than we were willing to pay for that feedback.

Early in my management career, I believed most processes failed because they insufficiently addressed the problem at hand. I thought that if a process' results were poor, it was because the process needed more nuance and sophistication. What I've learned since, mostly through designing thoughtful processes that nonetheless failed, is that a good process is a deliberate trade-off between quality and overhead. It's common to see a code review process that runs quickly but provides low-quality feedback (for example, requiring code reviews be completed within two working hours of a pull request being made). It's equally common to see a very involved, high-quality process that folks eventually ignore because it's too slow (as in the case of the DUCK Review).

To figure out the appropriate degree of process sophistication for your team, you have to start by figuring out how you'll run the processes and reason backward from that answer to determine the degree of process sophistication your organization is capable of sustaining. To help you think through that question, in this chapter I'll work through:

- Patterns for running processes and how companies typically progress through those patterns
- The pros and cons of these patterns
- How to operate the baseline pattern successfully
- How to deal with the budgeting realities
- How to navigate the trend cycle

When you've finished, you'll have a clear perspective on who should be running the process within your organization as it exists today, when you should consider switching patterns, and how to avoid the most frequent mistakes made in the staffing process.

Typical Pattern Progression

There are five frequent patterns I see organizations use to run their Engineering-scoped and company-scoped processes, and they tend to appear in the same sequence as I have listed them here. Whatever pattern you're using today, it's likely you used a different one three years ago, and if you're growing quickly then it's likely you will use a third pattern three years from now. There's no right pattern, just the most appropriate pattern for your current constraints. (However, some patterns *do* seem to go wrong more often than not.)

EARLY STARTUP

Brand new companies all use what I call the Early Startup pattern: either a founder or the functional executive does something for their function, or it doesn't happen. In my first year as Calm's CTO, there were no shared People processes, and I ran Engineering's performance review process out of a spreadsheet. It needed to get done and I was in the position to do it. This is how very small companies work, typically through their first 30–50 hires, at which point they move onto something slightly more structured.

BASELINE

The next phase is the Baseline pattern, which is first adopted around 50 hires and is the final pattern for most companies. Company-scoped processes (e.g., company planning, performance management, and budgeting) are run by a central People (or Human Resources) function and Engineering-scoped processes (e.g., the tech spec review, the incident review, and the developer productivity survey) are run by engineers and Engineering managers. There are no specialized roles to support Engineering processes, and limited specialization of the company's processes to support Engineering.

SPECIALIZED ENGINEERING ROLES

As companies grow beyond 200 engineers, Engineering-scoped processes often start to take enough time that they benefit from full-time support. This often happens first for either an incident review or tech spec review, but depending on your company's particulars, any process might be the first one to become sufficiently expensive to run. I call this the Specialized Engineering Roles pattern because most companies address this problem by spinning up a new function—usually either Technical Program Management or Engineering Operations.

COMPANY EMBEDDED ROLES

At many technology companies, Engineering is the largest function and consequently will reach certain points of scale earlier than other parts of the business. Ensuring that processes work properly despite all these extended requirements often leads to Engineering splitting away from the company's default mechanisms for hiring, performance management, and so on, such that there are dedicated recruiters and members of the People team working with Engineering. I call this the Company Embedded Roles pattern.

In addition to pure size, Engineering has several other factors that create a particularly complex set of requirements, and support the switch to this pattern:

- There is a dual-sided career ladder across engineers and Engineering managers, whereas other functions typically have only one.
- There are often one or more specialized roles along the lines of Developer Relations, Technical Program Management, Quality Engineering, or Security Engineering.

- Engineers and Engineering managers tend to be slower and more expensive to hire and retain than most other functions.
- A long ramp-up time, from starting to becoming highly productive, places a higher value on retention.

BUSINESS UNIT LOCAL

Finally, there is the Business Unit Local pattern, where Engineering reports into each business unit's leadership and there is no longer any one leader who is responsible for the entirety of Engineering. This typically results in Engineering moving back to a standardized process run by centralized functions, or embracing inconsistency as each business unit's Engineering organization finds its own path forward.

Patterns' Pros and Cons

Now that we've introduced the typical sequence that companies go through when implementing these process management patterns, I want to go a bit deeper into the pros and cons of the patterns themselves:

EARLY STARTUP

When you start a new company, it's just one or more founders. Early hires will almost always focus on building a small product to validate the market or run the business day to day, and generally won't include anyone focused purely on process. This makes sense, because processes are about coordinating large groups and the company is only a handful of people at this point. Any process that you want to run needs to be run by the founders, or the managers hired by those founders.

Companies usually exit this pattern because there is an impactful process they want to run—initially, this is often performance management—but they feel they're too busy to run it with the current team.

- Pros: Low cost and low overhead; there are few downsides at this small scale because most process is focused on supporting large teams.
- Cons: Often valuable stuff doesn't happen because everyone is too busy; sometimes the quality of a process is very low due to founders or managers having had limited time or experience running that process.

BASELINE

Engineering-scoped processes are run by engineers or Engineering managers, and company-scoped processes are run by a centralized function, usually in People or Human Resources. Most companies stay in the Baseline pattern for an extended period of time, with perhaps one small exception for a dedicated recruiter for Engineering hires. It typically works well through 100 or more engineers.

One of the reasons this works significantly better than the Early Startup pattern is that it identifies a clear team responsible for selecting vendors (e.g., Greenhouse for hiring needs, Lattice for performance management). These tools are not perfect, but they're significantly better than a spreadsheet, which is the default tool when seven managers are independently trying to run a decentralized process.

- Pros: Some modest specialization allows Engineering to focus on engineering rather than on foundational company processes; complexity of HR tasks tends to outstrip the Engineering managers' depth by this phase (e.g., managing visas); unified systems allow the executive team to inspect across functions rather than just being limited to their own function.
- Cons: Outcomes are highly dependent on the quality of these centralized functions; changing company-scoped processes becomes challenging.

SPECIALIZED ENGINEERING ROLES

The goal with this pattern is to increase efficiency in Engineering-scoped processes by hiring dedicated, specialized roles to fulfill that work. Typically, this means hiring either an Engineering Operations or Technical Program Management team. The first areas of investment are usually organizational planning, incident management, and engineering onboarding.

- Pros: Specialized roles generally introduce significant efficiency, offering a better skill set for the task to be done at a lower salary point; introducing these roles is usually energizing for the individuals no longer doing the work; specialists who are able to focus on a given process often make significant improvements in that process.

- Cons: While having specialized engineering roles is more efficient, it's usually more expensive as these roles introduce additional headcount. It's a lot of work to effectively support specialized roles like Engineering Operations or Technical Program Management teams. Specialized roles often freeze a company in a given way of working. Whereas engineers are incentivized to eliminate a process, the specialist is incentivized to improve it.

COMPANY EMBEDDED ROLES

When this pattern is used, Engineering's needs for company-scoped processes have reached a sufficient scale or have sufficiently diverged from standard operating procedure such that support functions like People and Recruiting hire individuals specifically to support Engineering. This usually happens one function at a time, often starting with Recruiting, followed by People and Finance.

While it will often happen first for Engineering, this pattern will repeat for other sizable organizations in the company. For example, you'll frequently see a setup where Engineering and Sales have dedicated embeds, and the rest of the organization works with a centralized team. Within a sufficiently large company, every function will have dedicated support.

- Pros: Company embedded roles empower Engineering to customize its process and approach, while building stronger relationships across functions.
- Cons: Expensive to operate and quality is heavily dependent on the embedded individuals.

BUSINESS UNIT LOCAL

With this pattern, each business unit in a company has its own separate Engineering function. This split typically occurs when an organization becomes large enough to have multiple meaningful business units, and there's a perception that standardization across Engineering in those business units is unnecessary or unnecessarily expensive. For example, one business unit might require HITRUST compliance, whereas the others do not, which might lead to significantly different engineering trade-offs.

Sometimes this pattern is adopted but one business unit is so much larger than the others, it becomes the de facto Engineering hub for the wider

company, including leading cross-organizational developer tooling and infrastructure investments.

- Pros: Aligns Engineering with business priorities within each business unit.
- Cons: Engineering processes and strategies often get stuck wherever they were when the split first happened; future changes require consensus across many Engineering leaders, and investments across Engineering become difficult to prioritize.

All of these patterns have flaws but are worth considering. That said, if you asked me to pick one without any additional context, I would always recommend the Baseline pattern. It's generally good enough and the easiest to manage.

Operating the Baseline Pattern

What I've called the Baseline pattern is the most common approach to running processes, with a centralized People organization running company-scoped processes and a mix of engineers and Engineering managers running the Engineering-scoped processes.

It's common for a handful of reasons:

It's the default scenario.
: Simply don't hire any specialized roles or embed support directly within Engineering and you're at the baseline.

It's relatively simple.
: You don't have to learn to hire or manage anything new, and it doesn't require treating Engineering differently from other functions.

It appears cheaper.
: When you propose moving to another pattern, it's usually framed as a way to free up existing capacity to focus on higher-value work, but that requires hiring additional roles to support the pattern, such as technical program managers. That makes it more expensive from a budgeting perspective, even if it's meaningfully more effective, and most budgeting discussions are cost-driven rather than effectiveness-driven.

There's a fourth reason it's common: It works pretty well! Every company I've worked at, including Uber and Stripe, got very far before leaving this pattern,

well past 100 engineers. I personally believe that many companies switch to other patterns too early, often because they're copying the playbook from their previous, much larger, employer.

If you're tempted to switch away from Baseline and are below 100 engineers, there are a few things I'd recommend focusing on first to try to make the current pattern work for you:

Rotate the work.
: Calm has had great success with six-month rotations, where individuals serve in one capacity (e.g., running incident review) for six months before swapping off. We increase continuity by having two people serve together in each capacity with overlapping but offset periods of service.

Make the process interesting for the people running it.
: If someone is leading your incident review, also give them exposure to the company's wider planning process as you factor in reliability projects. If someone is running your tech spec review, pull them into the annual revision of your Engineering strategy (Chapter 3). You should go beyond service; folks should feel that these are unique learning opportunities to work with senior leadership.

Ensure that your promotion rubrics, or your promotions if you don't have a rubric yet, value this kind of work.
: You can even experiment with the rubric, strongly preferring that engineers serve a term in one of your processes before moving into Staff-plus or managerial roles. People are smart and drift toward the work that you value. Make sure to call out the individuals doing the work behind your processes, particularly in all-hands and Q&A meetings (Chapter 10).

There are no guarantees, but my lived experience is that making these relatively minor tweaks will allow the Baseline pattern to function well and keep working much longer than you might expect.

Dealing with Budgeting Realities

While there are operational challenges to moving across patterns, most of those challenges are only obvious in hindsight. Instead, the biggest impediment for moving from Early Startup to Baseline or from Baseline to Specialized Engineering Roles will be the budget implications. Even if you can show that three technical program managers would significantly improve Engineering's

impact—which likely is true but you likely can't prove with a spreadsheet—most budgeting processes are anchored to relative dollars rather than absolute impact.

Digging in, the budgeting problems that arise from pattern shifts are:

Most companies that successfully shift up the patterns' cost curve do so when they are rapidly growing.

Such companies are comfortable with rapidly growing headcount costs and are generally tolerant of adding roles to support that growth. In all other circumstances, companies generally aren't tolerant of adding new roles and structures, even if they would likely increase efficiency. In the latter scenarios, you're much more likely to hear folks talk about "doing more with less" than "designing our organization to operate efficiently," and it's very hard to make efficiency-based arguments with an executive team that's in a cost-reduction mindset.

Many companies operate to a portfolio allocation target across departments.

For example, 20% may go to General & Administrative (G&A), 30% to Sales & Marketing (S&M), and 50% to Research & Development (R&D). Most pattern shifts will eat into the G&A budget, increasing G&A costs and preventing other planned G&A hiring. This generally means that many of your peer executives will be resistant to the proposal.

A surprising number of budgeting processes operate in headcount rather than budget.

This reality makes it hard to intuitively express ideas like "replacing two high-cost staff engineers with three mid-level technical program managers." Headcount-anchored budgeting processes portray the latter as more expensive than the former, even though it's not. It's possible to push through this confusion, but it imposes an ongoing communication cost.

Each of these concerns can be overcome with planning, patience, and conviction, but you have to decide yourself whether it makes sense to do so. My general advice is to stick with the Baseline pattern and avoid moving up the pattern cost curve unless revenue or headcount is growing significantly.

Navigating the Trend Cycle

There are numerous process trends. For example, the industry's era of Agile obsession: I once worked with someone whose job title was Agiletect. Similarly, there are also trends in who works on processes, and whether evolving processes is highly impactful, essential work or boring, dull, glue

work that won't get you promoted. In my own career, I've seen a major shift from "technical management" to "people-centric management," and an equally abrupt counter-reaction from "people-centric management" to "management is overhead."

If I bought into each of these trends, then I'd inflict a constant churn on the teams I lead, and we'd never get particularly good at any given way of working. That's not to say that trends are meaningless; there's always something real behind a given trend. In industry downturns, there will be a trend that reduces cash flow. In industry upturns, there will be a trend that takes advantage of copious investor cash. The shift toward people-centric management was in part driven by a generational shift in U.S. employees, with Millennials becoming a larger part of the workforce. There's something to learn from these trends, but it's a mistake to take them at face value: What has been working thus far hasn't become irrelevant overnight.

My advice is to design your organization around processes grounded in one specific set of foundational beliefs and stick solid to those beliefs for at least three to four years before changing them again. Change can be restorative, but if you change too quickly then you'll never be able to learn what is or isn't working well enough to inform future changes.

Summary

In this chapter, you reviewed the common approaches to running processes within Engineering and can now make an informed choice on which best suits your current needs. In particular, you've seen how many companies scale the Baseline pattern further than expected before adopting more complex models, and that approach may well work for your organization, too.

Find links to further reading and resources on *https://lethain.com/eeprimer-refs-19*.

Hiring

Everyone in an Engineering organization contributes to the hiring process. As an engineer, you may have taken pride in being an effective interviewer. As an Engineering manager, you may have prioritized becoming a strong closer, convincing candidates to join your team. As a more senior manager, you will have likely shifted focus to training others and spending time with candidates for particularly senior roles.

As an Engineering executive, your role in the hiring process will shift once again. You'll continue to make some key leadership hires yourself, but you'll spend more and more time designing and debugging your overall interview process.

In this chapter, we'll cover:

- Establishing your overall hiring process, including job descriptions, rubrics, and hiring loops
- How executives can focus so much on perfect hiring processes that their processes fail
- Your role in monitoring and debugging the hiring process
- Helping close key candidates across your organization
- Options for leveling candidates appropriately
- How managing headcount is a key part of managing hiring
- How to train your hiring managers to avoid challenges like pursuing non-existent unicorn candidates
- Calibrating hiring inside and outside of your network, along with considering internal candidates
- Evaluating building an Engineering brand for your company

- Deciding whether to introduce a hiring committee process
- Remembering that the system serves you, not the opposite

After reading, you'll have a clear plan for structuring your overall hiring process, as well as your specific role within that new process. You've spent much of your career serving a hiring process, and now you need to create a system that serves you.

Establish a Hiring Process

Unless you're joining an extremely early-stage company, your Engineering organization will already have some sort of hiring process in place. Unless there's widespread agreement that the current process isn't working, you should participate in the existing process to get a feel for how it works.

It's almost always the case that you can adapt the existing process to accomplish your goals rather than starting from scratch, and incorporating what already exists will both simplify retraining the team on the new process and build goodwill with the folks who built the previous process.

Regardless of where you start, your final process should include every one of these components:

Applicant Tracking Systems (ATS)

A good ATS is the core mechanism for coordinating your interviewing process. Although many early companies try, running an effective hiring process without an ATS is time-intensive with limited return: don't try it. There are enough reasonable options out there that I won't recommend any one in particular.

Interview definition and rubric

Define an explicit problem or set of questions to ask for each interview. Then add an explicit rubric for evaluating that candidate's answers. My experience is that it's preferable to be very consistent on which questions to ask. For example, using the same programming problem for all candidates' programming interviews.

A frequent pushback is that candidates will cheat by learning the problem from previous candidates, which is certainly possible. However, I've found the risk of cheating is still lower than the risk of poor signal due to solving inconsistent problems. (Furthermore, it's usually pretty clear which candidates are cheating. Make sure you have additional sections for them

to complete if they go fast and note if their ability to solve those sections degrades in a surprising way.)

Interview loop documentation
: Every role should have a documented interview loop that covers the interviews, identifies the trained interviewers for each interview, and links to each interview's definition and rubric.

Leveling framework
: Articulate how you level candidates based on their interview performance and prior experience. In particular, describe when you level candidates in your process.

Hiring role definitions
: Every interviewer, hiring manager, and recruiter will engage with your hiring process using assumptions built on the prior processes they've worked in. This will often lead to disagreement between hiring managers and recruiters about who's responsible for the closing process, who has input on the offer's compensation details, and so on. The best way to avoid this is being very explicit about who is responsible for what.

Job description template
: You should create a baseline template for job descriptions, with a consistent structure and background on your organization, benefits, and mission.

Job description library
: Hiring managers should use the job description template to write a job description for each role they hire. These job descriptions should be aggregated in a shared folder where they can be reused rather than reinvented. This also simplifies keeping descriptions updated as you refine shared components.

Hiring manager and interviewer training
: Finally, the last component of an effective hiring process is a clear mechanism for training your interviewers. The most effective process I have seen is having new interviewers shadow several trained interviewers, combined with one reverse-shadow interview where an experienced interviewer shadows (and gives feedback to) the new interviewer.

There are certainly other approaches to consider, including training materials or classes, but I've found that many interviewers simply don't listen in those

trainings, whereas shadowing and reverse-shadowing make it much harder to fake your participation.

If you're joining a relatively scaled Engineering organization, it's likely that most of these components will already exist, and that you can quickly formalize the few undocumented portions. On the other hand, if you're joining a smaller organization, it's quite possible that you'll start from a place where none of these materials exist. In the latter case, I'd aim to introduce one or two components at a time over the course of a year: Going too fast will overwhelm the team, but isolating changes will lead to retraining fatigue from the team as their hiring process changes repeatedly.

Pursue Effective Rather Than Perfect

The two biggest errors that executives make in designing their hiring processes are not designing a process at all—hopefully addressed by the preceding section—and designing overly heavy processes that make it ineffective to hire. The latter is particularly challenging to notice because you'll often believe you are optimizing the process when you're actually slowing it down.

The three clearest indications that you've over-optimized your hiring process are:

- Recruiters hire fewer than five engineers per recruiter per quarter (excluding the scenario where they are constrained by headcount).
- You frequently need new interview loops or new interview questions.
- It routinely takes more than two weeks from a candidate's first interview to you making an offer to that candidate.

Each of these indicate a process that's consuming a lot of energy without generating much impact. Often the cause is an indecisive executive who injects additional interviews hoping for clearer signal, which generally obscures reality rather than clarifies it. I've also seen this caused by well-meaning, structured thinkers who are trying to replace biased human judgment with more structured approaches. Neither of these are inherently bad ideas, and it's through inspecting the above indicators that you can check whether you're really improving your process or if it just feels like progress.

I recommend you require a high bar for each extension to your hiring process. Even if individually they make a great deal of sense, in aggregate they will quickly make the process too cumbersome to operate. Each specialized interview

loop, each additional interview to design and train the team on, each approval step, each movement from an accountable individual to a committee—all of these will improve quality, but often in a way that leads to worse outcomes as the process grows heavy. If the current process works, even if it's not ideal, push the team to work with it rather than extend it. You should certainly modify the process when it's wholly broken, or when you can improve the standard path for everyone, but stay wary of specializations, customization, and the bespoke.

Monitoring Hiring Progress and Problems

Once you've built the hiring process, your job as an executive is generally to monitor and debug it, rather than serve within it. This is particularly true after Engineering grows past 100 members, at which point you'll be directly involved in the process for a small fraction of the senior-most hires.

Here are the mechanisms I've found effective in monitoring and debugging hiring, in the order that I'd consider rolling them out if I joined a company without much oversight:

Include Recruiting in your weekly team meeting

Your Engineering leadership team should have a weekly team meeting (as discussed in Chapter 10), and I strongly encourage including a tech recruiter in that meeting. Their presence makes it possible to troubleshoot recruiting topics quickly and transparently. Some topics may not be particularly interesting to the recruiters, but that's true for some members of most standing meetings.

Your weekly team meeting with your full team and tech recruiter is by far the easiest place to change hiring priorities without anyone feeling left out of the loop. Almost any other meeting will be missing at least one highly impacted party.

Conduct a hiring review meeting

Meet once a month with the Engineering recruiting lead, and talk through their priorities, as well as any problems that have come up. Keeping this meeting small, typically just the two of you, means you can troubleshoot difficult issues that may be hard to discuss in your team meeting.

Maintain visibility into hiring approval

Although you should likely not be approving every hire in your organization, it's extremely valuable to have a single place where you can see all the approvals. Often this is a private chat with Engineering's hiring managers

and recruiters, where each offer is approved. It's worth acknowledging that your approach will likely shift between periods of high growth where you're hiring dozens of folks a quarter, and low growth where you're making a few hires a quarter—if the absolute number of hires is low, it's worth reading every packet to remain in the loop.

Approve out-of-band compensation
: This will be discussed in more depth shortly, but similarly to seeing all hiring approvals, it's even more helpful to be an approver on all atypical candidate offers. This gives you visibility into the places where your standard operating process isn't working for some reason.

Review monthly hiring statistics
: Have Recruiting report on hiring statistics for each role they're currently hiring. It's particularly helpful to understand throughput (hires per recruiter), time to hire, offer rate, and acceptance rate. Those four metrics should be enough for you to identify the next set of questions to dig into.

There are, of course, always more meetings and tools that you can introduce. I'd recommend starting with a couple and going from there. As you've probably picked up by now, my experience is generally that you can go faster by making incremental changes than by introducing massive changes, even when your goal is transformation.

Helping Close Key Candidates

Sometimes executives insert themselves as a final interview in the hiring process, even after their organization becomes quite large. Executives who do this tend to swear by it as an essential step, where only they can ensure the quality bar for hires remains exceptionally high. However, it tends to significantly slow down the hiring process, and even executives who believe in it the most strongly will eventually scale back on this practice.

While it's unscalable to remain as an interviewer across all loops, it is particularly valuable to remain engaged in helping close senior candidates. As an executive, you should be able to tell Engineering's story and how it contributes to the larger company story, and why that makes for interesting work. You're also best placed to address any strategic concerns the candidate raises.

The approach I've found helpful here is three-fold:

1. Let the Recruiting team know that you're available for sell calls with senior candidates.
2. Update the Staff-plus and Engineering manager hiring loops to offer you as a sell call *by default* for candidates who ask for it. Many candidates won't, but some will, and these are the candidates who will shape your company the most quickly after they're hired.
3. Insert yourself into the selling process for particularly high-importance roles. Even if you don't say something new, often the same message from an executive will carry more weight and your involvement is a clear signal to the candidate that they're considered an important hire.

I've only found this counterproductive in two scenarios. First, some executives add a sell call as a mandatory part of the hiring process, which often creates more friction than it's worth. There are candidates who are excited to accept without meeting you, and for them the additional sell call will slow things down—executives are particularly painful to schedule. Second, there are some executives who are exceptionally bad at selling their organization. A friend once did a sell call with an executive who turned out to be watching online videos during the call, which unsurprisingly did not make them feel like a valued candidate.

Leveling Candidates

Within the hiring process, the two most contentious topics tend to be determining compensation details for each candidate and determining the candidate's level. You can't determine appropriate compensation for a candidate without knowing their level, so we'll start there.

The first question to answer is when you level candidates in your process. The *obvious* answer is that you level candidates after seeing their interview performance, but there are a few issues with that. Most importantly, you likely want to conduct a different process to evaluate very senior candidates than to evaluate early-career candidates. At a minimum, you'd want the interviewers to be different, as it's relatively rare for a panel of mid-level interviewers to decide a candidate is Staff-plus, and you likely wouldn't be confident in their evaluation even if they did.

Generally, I recommend provisionally leveling candidates before they start the bulk of your interview process. For example, you might provisionally level them after they complete the technical phone screen, allowing you to customize the remainder of their process for that provisional level. You can then finalize the leveling decision as part of deciding whether to make an offer. I recommend relying on a simple provisional leveling heuristic, such as a combination of technical phone screen performance and years of prior experience. This is far from perfect, but there's generally enough signal there to determine the range of plausible levels.

The final leveling decision should be guided by a written leveling framework, which looks at the candidate's holistic interview performance to determine a level. Part of that framework is handling disagreement around leveling, which is particularly common. The most common approach is that:

- A final leveling decision is made by the hiring manager, then escalated for approval.
- Approval is done by the hiring manager's manager for levels at or below your career level (also known as "terminal level" at some companies, this is the senior software engineer role at many companies), and approval for more senior levels is done by the Engineering executive.

Some companies, particularly larger ones, rely on a committee rather than individual hiring managers for these decisions. My experience is that committees appear less biased, but generally introduce a bias of their own, and are less efficient than wholly accountable individuals. The counter-balance is that at a certain scale, it's simply impossible to centralize these decisions without significantly slowing down your hiring process. I recommend introducing committees only after relying on individuals has proven too slow for your rate of hiring.

Determining Compensation Details

Compensation is a broad topic, which is covered in Chapter 22 in more detail, but it's worth a quick overview here on determining compensation for your offers. There are two particularly important questions that should be detailed in your hiring role definitions: who calculates the initial offer, and what are the approval steps once an offer has been calculated?

The approach that I've found effective is:

1. The recruiter calculates the offer and shares it into a private chat channel with the hiring manager.
2. Offer approvers are added to the channel to okay the decision to make an offer, the candidate's leveling, and the offer details.
3. Offers following standard guidelines; e.g., at or below 1.0 compa-ratio (an individual's current compensation divided by the target market rate for their position, discussed in Chapter 22) require approval from the hiring manager's manager, and those outside the guidelines require approval from both the hiring manager's manager and the Engineering executive.
4. Any escalations or modifications to the offer occur within the same private chat, to ensure all relevant parties are aware.

A centralized approach, where recruiters follow a structured process to calculate offers, has many benefits. First, it facilitates training on the shared process, and retraining on that process as it evolves. Second, it avoids less effective hiring managers leaning on compensation, such that those hired by worse hiring managers get outsized compensation packages (which is surprisingly common, although of course it's almost always framed as the weak hiring manager pursuing exceptional candidates). Finally, you can still design a process to make offers outside your compensation bands if you want to, but it should have a centralized mechanism to make it easier to both manage costs and drive consistency.

Some managers argue that this approach doesn't give them enough flexibility to make compelling offers to the best candidates. That's true, but I've consistently found that there's always another way to close a candidate other than more compensation. Further, outsized compensation packages will *always* create ongoing problems in your annual compensation process, which will be designed to normalize compensation across individuals with similar performance ratings at a similar level. Your broader perspective as the Engineering executive is necessary to balance these incentives, whereas an individual hiring manager is almost always incentivized to hire even if it creates a long-term mess for the wider organization.

Managing Hiring Prioritization

The intersection of headcount planning and hiring is discussed in Chapter 4 but is worth mentioning here as well. In practice, there are two fundamental modes of prioritizing hires:

- In rapidly growing companies, there is so much headcount that you'll be constrained by recruiter bandwidth rather than headcount.
- At slower growing companies, headcount is the more likely constraint.

In both cases, you'll frequently have teams pushing for higher priority for their roles. I'm a believer in forcing leaders to solve within their constraints rather than frequently shifting those constraints, but my preference is just one of many ways to approach these trade-offs. The most important thing to highlight is that both recruiter assignment and headcount are global constraints that you must control as the Engineering executive.

This control can either be something you do personally or something you delegate to one individual to do on behalf of the wider organization, but the decisions must be made centrally. Making these decisions centrally doesn't mean that you have to spend a lot of time on it. The simplest way to sidestep this is to determine the headcount and recruiters for each Engineering sub-organization (roughly, each area corresponding to one of your direct reports) and then allow those sub-organizations to optimize within their boundaries and allocations.

The biggest trap to avoid is prioritizing recruiters based on hiring needs. This often steers all recruiting capacity toward your least-effective hiring managers. My learned belief is that slow hiring is almost always an execution issue by the hiring manager or the recruiter, and only infrequently the consequence of limited staffing. The exception is when you've opened too many concurrent roles for your current recruiter staff, which is easy to diagnose by looking at the ratio of recruiters to roles (if you have more than three open roles per recruiter, something is very likely going wrong). If you really want to help, first consider spending time training the individuals involved rather than shifting headcount or recruiter staffing.

Training Hiring Managers

Earlier, I mentioned shadowing and reverse-shadowing as an effective mechanism to train interviewers. That is a crucial part of an effective hiring process, but there's a second component of training that's often ignored: training your hiring managers.

There's a handful of particularly common hiring problems that are usually due to untrained or inexperienced hiring managers. These problems include:

- Demanding unrealistic, unicorn candidates from the recruiting team, such that you never make offers.
- Allowing any concern from any interviewer to block a candidate from getting an offer, such that you never make offers.
- Pushing for non-standard compensation on every candidate rather than learning to close candidates within the standard compensation bands.
- Being indecisive on candidates and asking for additional interviews until the candidate withdraws from the process.
- Refusing to talk to candidates early in the process and then blaming the recruiting team that senior candidates aren't interested in finishing the process.

If you identify one of these, then I *do* recommend running focused trainings for your hiring managers on the specific topic that's coming up. These are all topics that I've devoted a session of my Engineering managers' monthly meeting to—talking through examples of why it's problematic and why it's not a sign of strong hiring, explaining what reasonable pass-through rates look like for a healthy hiring loop, and recommending strategies for overcoming the issue.

Once you've done a training session, you and the recruiting team should point out the issue to individuals who are running into it, and hold them accountable for fixing it. Folks making these mistakes will often have conviction that they're doing the right thing, but don't be swayed by their conviction. Effective hiring processes hire candidates. Hiring managers are accountable for their hiring process. Any argument to the contrary is a flawed argument.

Hiring Internally and Within Your Network

When I worked at Yahoo!, our team needed another Engineering manager. We didn't run a hiring process, or even do interviews. Instead, our director brought on a colleague he'd worked with before. That new manager soon decided he needed a tech lead on his team. We didn't run a hiring process, do interviews, or consider candidates on the existing team. Instead, our new manager brought over one of his previous colleagues. A third previous colleague reached out to our director, and without a single interview we'd soon hired a new chief architect who would ultimately never write or read a technical specification about our product, nor contribute a single line of code.

One of my teammates—one who had joined the team through the more traditional route of interviewing—described this pattern as the flying wedge, and it's emblematic of the worst sort of network hiring. Hiring exclusively from your network will convince your existing team that they and their networks aren't wanted for important roles at your company. You can reduce this impression by limiting your involvement when you do recruit from your network (e.g., staying at a distance and not leaning too heavily on the hiring decision).

A similar, somewhat common, scenario is one where your company exclusively fills important roles with external hires. Each individual external hire may make sense but in aggregate, the pattern of prioritizing external hires will encourage your team to seek career advancement elsewhere, draining your organization of context and continuity.

When it comes to internal or external hiring, and hiring within or outside of your network, the ideal path is always moderation. Hire some internally, some externally, some from within and some from without. Too much reliance on a single path will either isolate your culture from allowing valuable opportunities to evolve, consolidate it into the culture that worked at a former employer, or prevent it from coalescing to begin with.

In my experience, almost everyone agrees with the preceding statement, but quite a few don't follow its advice. As I've dug into that, it's generally because of a missing hiring skill that's either of the following:

- An inability to hire outside their network (or within it), or
- An inability to *fairly* evaluate an internal candidate (or external candidate)

Periodically, look at the number of internal versus external hires for senior roles within your organization and dig into areas where there are exclusively hires of one sort. If you find a lopsided pocket of your organization, talk with the relevant leader and push them to make one hire of the sort they're currently ignoring. Even one such hire will force them to acknowledge the skill gap and start the process of fixing the imbalance.

If the person with a significant imbalance is you, then take it seriously! Don't hide from it by justifying the imbalance with philosophical or intellectual rationales. Instead push yourself to make one hire of the other sort. Particularly for new executives, I often find there's an underlying belief that they cannot close strong external candidates, and disproving that belief is an important part of your personal growth.

Increasing Diversity with Hiring

Many Engineering organizations are concerned about representation within their organization. If they start an initiative to improve representation, a common first step is reviewing the organization's composition, looking at the composition of recent hires, and setting targets for both measures.

If this is your first experiment with setting goals for representation, there is one antipattern that I'd encourage you to keep in mind: wholly offloading accountability for diversity to Recruiting. It's very common for these programs to focus entirely on representation of newly hired employees, but it's at least equally important to track retention and promotion of the employees you've already hired. Indeed, there are a number of companies where hiring ratios are quite good, but their organization's ratios are rather poor, and this poor outcome is driven exclusively by inability to retain diverse talent.

Having seen this exclusive emphasis on hiring and studied ignorance of retention, I've come to believe that most organizations somewhat deliberately—without ever acknowledging it out loud, and often without recognizing it themselves—design systems where the Recruiting team is accountable for diversity, rather than Engineering management. This may look like progress, but long-term success depends on being directly accountable for your organization's health, not offloading it to an organization with little influence on how your team operates.

Building an Engineering Brand

Rather than digging back into the details of building an Engineering brand (which are discussed in Chapter 12), I'll briefly repeat the key takeaways:

- Some companies invest heavily in their Engineering brand, and those companies will generally tell you that their branding efforts are a fundamental part of their success.
- However, most companies, including some very successful ones, do very little Engineering branding and rarely find themselves stymied by its absence. This doesn't mean you shouldn't do any, but rather that you shouldn't view it as a fundamental necessity.
- If you choose to invest in Engineering branding, a small investment can capture most of the upside. The biggest exception is for companies that sell to engineers in addition to hiring them; in this case, you may prioritize a larger brand effort from a lead generation perspective.

In general, if you're already finding enough top-of-funnel candidates for your hiring process, don't spend more time on this unless you can connect that time to another business objective, or have internal folks who find this work energizing enough to take it on as a side project.

Should You Introduce a Hiring Committee?

Many companies introduce centralized (Engineering-scoped) or semi-centralized (Product Engineering or Infrastructure Engineering-scoped) hiring committees as part of maintaining a consistent hiring process. I've seen this happen frequently enough in Silicon Valley companies that some executives have come to believe that hiring committees are a natural or ideal landing spot.

Hiring committees are a useful tool, but I'd caution against introducing them as the *obvious* solution. They're useful, but they come with their own problems.

I generally dislike committees as they introduce ambiguity around who should be held accountable for outcomes. Committees also make hiring decisions further away from the team the individual will join, which often degrades individual decisions. The biggest issue, though, is that committees are vulnerable to persistently misaligned members. I was once on a hiring committee that a new member joined who relied very heavily on the universities that candidates attended, even when we clarified to the member that we didn't hire that way.

They refused to change, and our Engineering executive was unwilling to hold them accountable to our hiring practices.

On the positive side, hiring committees are a great mechanism for training hiring managers' judgment on what makes a good candidate. They also introduce more consistent hiring practices across an inconsistent organization, solving a problem similar to the one addressed by Amazon's Bar Raiser program. Committees are slower than a responsive hiring manager, but faster than a disengaged or very busy hiring manager.

Remember That the System Exists to Support You

If you came up as a rules-minded leader, you can almost certainly think of times when the executive responsible for designing your hiring process also ignored that process to accomplish an immediate goal. Personally, I've been most annoyed by executives who steamrolled the process to hire former colleagues who performed poorly in our interview process. Each time I'd complain to colleagues, "Why did we build this comprehensive hiring process if we don't even trust its decisions?"

As is often the case, as I switched into the role of the executive responsible for Engineering hiring, I began to appreciate why perfectly following the process was difficult. When I vowed to loyally follow the hiring bands, sometimes I'd find peer executives paying far outside the bands, implicitly penalizing hires in Engineering. When I endeavored to respect each negative hiring review, sometimes I'd encounter interviewers who refused to use the stated rubrics. When I hired for a brand new role, I'd sometimes find interviewers who interpreted the role's requirements very differently, even when I pulled together materials that explained it.

Each of those challenges can be solved over time with better training, but as an executive you rarely control the timeline you're working in. Sometimes your problem is urgent today. In those scenarios, the question to answer is sometimes whether the company will be better off if you solve the underlying problem (e.g., missing a leader for a key role) or if you respect the process (e.g., don't break the rules you created). You should try to solve your problem within the process you've designed, but don't get so blinded by your process that you think the process is always more important than your problem at hand. Sometimes the process is clearly less important than the current problem.

That doesn't mean you should always ignore your process. When our hiring processes decline a candidate, that's usually something to pay attention to. Even

if we're confident the negative signals are wrong and we decide to hire that person, the hire is undermined when their new colleagues know they performed poorly in the hiring process. There is a cost to defying your process, just as there is a cost to following it, and as an executive you need to make that trade-off deliberately.

Summary

This chapter has covered your role as an executive in your organization's hiring and the components you need in order to build an effective hiring process. It has provided concrete recommendations for navigating the many challenges that you're likely to run into while operating the hiring process. There are an infinite number of questions to dig into, but with the guidance of this chapter, you are ready to get started, build a system that supports your goals, and begin evolving it into something exceptionally useful.

Find links to further reading and resources on *https://lethain.com/eeprimer-refs-20*.

Engineering Onboarding

Most companies say that it takes three to six months for newly hired engineers to fully ramp up. Engineering leaders know it's unwise to admit that it takes their team longer than that to onboard new engineers, so that's what they say out loud, but they generally believe it takes longer for a new engineer to become fully productive. They also know that their most impactful engineers are still becoming more productive after years with the company.

Running Engineering onboarding means optimizing two closely related goals:

- Increasing the percentage of engineers who are reasonably productive in their first three months
- Setting the foundation that supports engineers to become more extremely impactful within a few years from now

Done well, onboarding should excite new hires and raise the floor for success. Indeed, in rapidly hiring companies, effective onboarding is the highest-value investment you can make into Engineering productivity, but somehow it's often dismissed as a secondary concern. Fortunately, like many oft-forgotten processes, you can go from no onboarding to rather good onboarding in short order.

To get there, in this chapter we'll walk through:

- The fundamental components of onboarding, including examples of several real companies' onboarding processes
- The role of the executive sponsor, orchestrator, manager, and buddy in a typical onboarding process

- The curriculum to consider including in your onboarding process
- Why onboarding programs fail
- Whether to integrate with wider company onboarding
- When to prioritize onboarding

By the end, you'll have a plan for incrementally improving your current onboarding and a clear perspective on when to move from ad hoc efforts to a structured onboarding program.

Real-world Examples

What onboarding specifically looks like varies a bit across implementations, so I think it's helpful to start with a few concrete examples of real-world Engineering onboarding. There are two caveats to keep in mind as we talk through these particulars. First, these often omit significant efforts around company-wide onboarding, to focus specifically on Engineering onboarding. Second, every company's onboarding process evolves over time; any static description can only capture one specific iteration, and all of the companies discussed here have evolved their onboarding beyond the specific approaches described.

Examples of Engineering onboarding programs are:

A startup with 30 engineers

When I showed up at Digg, I got a company-branded t-shirt and a laptop. My manager introduced me to the two other members of my team. They tried to issue me an integration environment, but the one engineer who knew how that worked had recently quit, so they told me to take over another recently departed engineer's environment instead. With that, the onboarding was complete.

It's a bit messy, but a surprisingly large number of startups have effectively no onboarding beyond showing up, getting a laptop, and getting pointed to teammates who can help.

A startup with 150 engineers

New hires at Stripe spent their first day focused on laptop setup, security practices, and getting familiar with development tools by making a small commit to the product website (eventually adding oneself to the company website's About page). New Engineering hires were then run through a curriculum ranging from architecture to observability.

The final step of Engineering onboarding grouped new hires into teams of four to six engineers. Each team worked on a project together, with one existing Stripe engineer serving as the project's tech lead. These projects generally lasted one to four weeks and were completed before releasing the hires to join their long-term teams.

A company with thousands of engineers

Facebook runs a six-week Engineering Bootcamp attended by every new Engineering and Engineering management hire. It focuses on facilitating team selection between new hires and Facebook's teams, and teaching new hires Facebook's approach to development. They work directly on Facebook's software, often fixing bugs, and are supported by dedicated Facebook engineers who review their pull requests, run office hours, and so on.

Facebook's Bootcamp is the most frequently mentioned model of high-investment Engineering onboarding. If you are considering copying components of it, keep in mind both the volume of engineers going through Bootcamp (10s and later 100s of concurrent new engineers) and that its goals are both onboarding and team matching, something that is atypical of most companies' hiring processes.

Most companies with more than 50 engineers develop their own approach to onboarding that echoes one of these processes but has a number of unique details. These programs also tend to change a great deal over time. Many small companies try to copy Facebook's Bootcamp, but few continue copying it after they understand the sheer magnitude of the undertaking.

Onboarding Fundamentals

A structured onboarding process comes down to staffing a handful of roles (especially your role as the executive sponsor and someone to orchestrate onboarding) and deciding on the specific curriculum to include in your program. Let's get started by looking at roles.

ROLES

It's easy to put a lot of energy into doing one-off designs for your onboarding process, but onboarding programs are living things that require continual care. It's more helpful to start by thinking about the roles necessary to run the whole

process, particularly the executive sponsor, program orchestrator, team manager, and onboarding buddy.

Executive sponsor

As the Engineering executive, you probably won't run Engineering onboarding. You may lead an onboarding session for each class of new hires, which lets them know that you're a real person and builds a connection they can use to reach out if they have concerns later. However, you do have a very important role to play in onboarding, which is serving as onboarding's executive sponsor.

As onboarding's executive sponsor, you should:

- Select the program orchestrator to operate and evolve the program.
- Align with the orchestrator on how to monitor the program over time (certainly through surveys, but ideally through some objective data, too).
- Monitor the program data on a monthly basis (e.g., look for quantitative feedback from session attendees, or the number of sessions given).
- Meet directly with new hires who have exceptionally good or bad onboarding experiences to understand where things might be falling through the cracks.
- Celebrate the program on an ongoing basis so that people continue to prioritize its success; this includes supporting onboarding work as a valid contribution toward being promoted.

Amplifying the last point, companies struggle to prioritize onboarding, even when they know intellectually that it's very important. Problems can be easy to miss, because new hires are unlikely to complain about onboarding.

Only you, as the executive sponsor for Engineering onboarding, have the vantage point and authority to maintain an ongoing quality bar for an impactful onboarding program.

Program orchestrator

Where the executive sponsor provides the organizational resources to run an onboarding program, it's the program orchestrator who translates those resources into a living program. Most frequently, this is an Engineering manager who runs onboarding in addition to other responsibilities, but at a scaled company this might well be someone in Engineering operations or a technical program manager (covered in Chapter 19).

The general expectations of the program orchestrator are:

- Develop and maintain the program's curriculum (e.g., what content should be prioritized and how many courses should you have?).
- Evolve the program incrementally over time based on participant feedback and outcomes.
- Coordinate each onboarding class to ensure it runs effectively.
- Create onboarding templates that fit the curriculum and are used by each new hire's manager to continue their onboarding beyond the program itself.
- Align with the executive sponsor on monitoring the program's outcomes. (This should typically include a post-onboarding survey to all participants, a summary of survey responses, and data trends over time. You may also want to look at other developer-efficiency metrics, discussed in Chapter 6, to understand where to focus subsequent onboarding investments.)
- Address the little mishaps that naturally happen in a program like this (e.g., a segment's teacher calling out sick for the day).

Selecting an effective program orchestrator is the single most important decision you'll make for an effective onboarding program, and you'll make it over and over again as folks shift out of this role. In my experience, the most important qualifications are:

- Having real experience operating as an engineer at your company (or relationships to engage with those who have that experience)
- Being energized by people as individuals rather than being compliant with a process
- Being able to use feedback to drive ongoing improvement

Being the onboarding orchestrator is generally a part-time role until a company gets rather large (20 or more engineers joining per month), but it's not a small task. Whoever runs this program should be able to devote at least 20 hours a month to its success. An Engineering manager could take this on and manage a team, but it wouldn't leave them time for much else.

Team manager

Even the best onboarding process will have gaps and will prioritize the standard case over any particular individual, which is where the new hire's manager comes into play. While onboarding is focused on getting new hires off to a productive start, their manager is ultimately responsible for the new hire's long-term success at the company.

The general expectations of the team manager are:

- Create a personalized onboarding document for each new hire, starting from a template created by the program orchestrator.
- Start weekly 1:1s with the new hire. (Even if you prefer less frequent 1:1s, it's very valuable to have them weekly for a new hire's first several months.)
- Assign an onboarding buddy to the new hire. (In some cases, this is handled by the program orchestrator, in which case you should ensure an appropriate buddy is assigned.)
- Partner with the onboarding buddy to identify a new hire's first several projects. (Finding safe learning projects that also provide real value to the business is an art and will significantly accelerate the new hire's impact and sense of belonging.)
- Make sure the new hire has actually been added to the various team meetings, chat rooms, email groups, and so on. (Even if it's supposed to happen automatically, this stuff gets routinely messed up, and can be very demoralizing for new hires when it happens.)

Note

It's worth acknowledging that managers are also operating under their own pressures (for example, shipping a delayed project or restaffing a small production on-call rotation). It's surprisingly common to see some friction between managers' short-term goals and the onboarding program's longer-term goals (e.g., asking to pull their new hire from onboarding to task them on a key deadline). The program orchestrator should fight against this tendency by frequently reminding managers how important onboarding is to their team's success.

Onboarding buddy

The last important onboarding role is the onboarding buddy. The particulars vary a bit by company, but in most cases they are a second point of contact within the new hire's team. The buddy can answer specific questions from the new hire such as writing software on their team, setting up the development environment, and working through their initial project. A buddy is also a fallback for when the new hire's manager might be unavailable or slower to respond.

The general expectations for the ongoing buddy are:

- Bring the new hire along with you to meetings, to grab lunch, and to meet other members of the company. Don't let them linger alone! This is especially true in remote companies, where a bit of effort will transform the company from an anonymous wall of faces into a community.
- Schedule daily time with the new hire for their first several weeks. This can be 15 or 30 minutes with some time blocked afterward in case it runs long. The goal is to resolve any blocking issues or topics for the new hire. Having something scheduled helps minimize the common scenario where the new hire feels awkward asking for help directly.
- After one to two weeks, chat with the new hire about whether it's helpful to continue the meetings for another month or two. This will come down to their personality and preferences more than any sort of best practice.
- Keep in touch with the new hire's manager about areas where things are going well or where they're running into problems. It can be tempting to ignore early problems, but this is really the only chance you'll have to address behavioral and cultural misalignment before they calcify.

CURRICULUM

Once you've identified the orchestrator for your onboarding program, there's still the matter of deciding on your curriculum—what should you actually focus on teaching? As an initial point to experiment from, I'd recommend three segments:

- Engineering values and strategy
- Technical architecture
- Spinning up the development environment

Once you have those three overview courses, work on strengthening the curriculum from two angles:

- Survey new hires a month after onboarding to ask what they wish had been covered.
- Survey managers and tech leads about areas where new hires are struggling. These might be technical but are even more likely to be cultural.

This will steer you down a valuable path and keep you from getting lost in the endless potential onboarding topics. In particular, it will help you avoid including topics that are theoretically useful but that no one shows much real interest in.

The other question to consider is whether to include a project component in your onboarding program, like Facebook's programming components of Bootcamp or Stripe's onboarding project teams. The general answer here is that these projects work very well if you put a lot of energy into them, and otherwise don't go particularly well. It's worth keeping in mind, the alternative to running an onboarding-driven project is that teams select projects for their new hires, which usually go well and don't require centralized coordination. This isn't particularly structured, but it is what many companies with fewer than 50 engineers do by default.

Finally, one implicit topic in your onboarding curriculum is getting to know the people who joined the company around the same time. When you're looking purely at learning and outcome metrics, it's easy to miss what new employees often mention as onboarding's biggest impact to them: getting to know the group of other new hires and building a network of people to ask questions of and eat lunch with.

WHO CAN ATTEND ENGINEERING ONBOARDING?

The question of who is required to attend Engineering onboarding, as well as who is allowed to attend, will come up periodically over the course of any program. My recommendation is to require all engineers and all Engineering managers to attend, regardless of level. Some very experienced engineers and managers will attempt to skip, but I think that's a bad thing to allow. Onboarding is as much about why things work the way they do as it is about how they work, and the number one risk for senior hires is that they try to change how things work before understanding why they work the way they do. You should especially require attendance for senior hires who insist they don't need to attend.

If other functions, like Product, ask to attend Engineering onboarding, my advice would be to not initially schedule them to attend, but let them individually ask to be added to future onboarding cycles. This ensures that folks who really want to attend can, without requiring you to rework the Engineering onboarding to be a rewarding experience for those without an Engineering background.

Why Onboarding Programs Fail

It's tempting to think that onboarding programs fail because no one tried hard enough to create a great program. Certainly, some companies don't invest in onboarding, which is a lost opportunity, but that isn't what leads to failure. Instead, most programs fail because someone—with extremely good intentions—creates a complex, involved program.

The troubling sequence goes like this:

1. A new program orchestrator is selected to run Engineering onboarding.
2. They develop a new, more involved program that addresses all the identified needs, although it happens to take a lot of energy to operate.
3. The new program launches, and everyone loves it. Finally, onboarding is solved!
4. Over time, it's hard to get enough internal engagement to run the program. People still love the program, but they have too many conflicting priorities.
5. The orchestrator gets frustrated and transitions to another role. The program is run by their replacement, who often doesn't have the original orchestrator's passion for the area.
6. The program becomes stale and bureaucratic.
7. The program is scaled back significantly to something much simpler.

When you consider revising your onboarding process, make sure it's something the orchestrator can run for the long term. This needs to be something you can run when the excitement of a newly launched program dies down, and Engineering managers stop sending the orchestrator "thank you" cards. The worst onboarding programs come from trying too hard, rather than not trying hard enough.

Integrating with Company Onboarding

In addition to Engineering onboarding, it's likely that your company has a wider onboarding program. This program might touch on many of the topics that are important within Engineering onboarding, particularly those like users, annual goals, or company values. You should absolutely lean on the content from company-wide onboarding segments and avoid duplicating existing materials.

A controversial question that often comes up is whether the team coordinating company onboarding should also coordinate Engineering onboarding. For example, if the People Programs team is running onboarding, shouldn't they also schedule the onboarding sessions for the Engineering team?

My experience is that you generally should continue running onboarding from within Engineering. There are a few reasons for this:

- There is usually significant support for handing off the program to a centralized function to run it, because the program orchestrator sees it as an opportunity to take on a new role. However, the people who replace the program orchestrator will be less focused on effective Engineering onboarding and will not have experience operating as an engineer at your company. They will approach Engineering onboarding with a focus on making it more consistent with the shared company onboarding process, even if that means eliminating key aspects of what makes Engineering onboarding effective. This isn't because they're bad people, it's just the natural incentives of a centralized function.
- Once another function (such as HR) is responsible for running your onboarding process, experimenting with improvements will become much more difficult, as it requires cross-functional coordination with a team that is often very busy. Increasing the cost of experimentation will significantly harm onboarding quality over time.
- Feedback on the quality of onboarding will often become sensitive data within the other function, perhaps shared with you infrequently and only after being sanitized. Rather than being able to drive your onboarding with data, including knowing who to reach out to about problems, you may lose access to that data.

Sometimes circumstances will align such that it's unduly painful to maintain control of Engineering's onboarding process, and that's fine. Sometimes ownership of these problems becomes sufficiently political that it's difficult to retain

ownership; executive leadership is about taking the best available path, not dying on every sword that presents itself.

When to Prioritize Onboarding

Onboarding is particularly impactful when you're hiring a solid chunk of new people every month. If you have 10 new hires each month, then the economies of running classes will start to outpace one-on-one training. This is doubly true when you begin accelerating hiring, as your existing team will be both busy hiring and out of practice at onboarding new engineers.

Even if you're hiring modestly, good onboarding is a valuable investment. Effectively ramping up new hires is always worthwhile. The only case where I'd recommend against investing in onboarding is when you're hiring so infrequently that onboarding materials go out-of-date before they're used a second time. In those cases, it's better to invest in personalized onboarding with the individual it is to than design a larger-scale program. Keep in mind that the decision to prioritize onboarding is not only a matter of headcount growth, but also of your natural attrition rates. Many companies are not growing their Engineering organization, but still hiring many new team members due to attrition.

Finally, remember to experiment your way into the right level of onboarding. Start with something simple that you can pull together in a few days, and that might be enough.

Summary

In this chapter, you've learned about the wide range of approaches that companies take to onboard engineers, and why onboarding is as important as hiring to running an effective organization. If you, like many organizations, currently have a whole lot of nothing as your onboarding program, get started with a small iterative approach as you evolve toward something exceptional, and remember that good onboarding is exceptionally valuable but it absolutely is not free.

Find links to further reading and resources on *https://lethain.com/eeprimer-refs-21*.

Performance and Compensation

Uber's original performance review process was called "T3B3" and was remarkably simple: write the individual's top three strengths and top three weaknesses, and share the feedback with them directly in person. There was a prolonged fight against even documenting the feedback, because they worried that writing the feedback down—where it might later be discovered by others—would discourage honesty. On the other side of things, there are numerous stories of Google employees spending months crafting promotion packets that ultimately don't get them promotions. Among those who've worked within both Uber and Google's promotion processes, there are advocates and detractors, and absolutely no consensus on what an ideal performance review process looks like.

Compensation poses a subtly different set of problems, but similarly there are no universally appreciated compensation review processes out there. Highly structured, centrally orchestrated compensation systems often converge on folks who, at a given level, receive similar compensation, even if their impact is quite different. More dynamic compensation systems disproportionately reward top performers, which introduces room for bias.

Because there's no agreement on what performance or compensation process you should use, you'll likely end up working within a variety of processes. This chapter digs into:

- The conflicting goals between those designing, operating, and participating in performance and compensation processes
- How to run performance review processes, including calibrations, and how to navigate their challenges
- How to participate in a compensation review process effectively

- How often you should run performance and compensation cycles
- Why your goal should be to run an *effective* process rather than a perfect one

Every one of these processes is loaded with trade-offs and traps that you'll need to be wary of, and after finishing this chapter, you will be prepared to plot the right course for your organization.

Conflicting Goals

Going back to Uber's T3B3 performance process, which identified an employee's top and bottom three areas for a given six-month period, what was most remarkable was its radical simplicity. It was focused exclusively on providing useful feedback to the recipient. To this day, I find that clarity of purpose not only impressive, but also rare.

Most performance and compensation processes have far less clarity of purpose because they try to balance many priorities from many stakeholders. A typical process at a given company tries to balance all of these distinct stakeholders:

Individuals
: They want to get useful feedback so they can grow. They want to get promoted and maximize their compensation, and they want to do so as soon as possible.

Managers
: They want to provide fair and useful feedback to their team and promote their team as appropriate (and in alignment with the various commitments they've made to their team). They want to provide appropriate compensation (and, once again, do so in alignment with the various commitments they've made to their team). Oh, and they also want to do all of this quickly!

People (or Human Resources) team
: They want to ensure individuals receive valuable feedback and create a "floor" for the quality of that feedback. They want to document feedback so that it can be used in performance management and legal scenarios later, to support the company's perspective. They want to be able to demonstrate the performance process for compliance purposes (e.g., SOC 2 requires annual performance reviews) and create structured input to calculating compensation.

Executives

They need to decide who to promote based on inconsistent evaluations across managers. They want to optimize the allocation of a fixed compensation budget to meet organizational objectives, and minimize the impact of inexperienced and misaligned managers on promotions and compensation.

I've never encountered, or heard of, a process that solves *all* of these stakeholders' needs elegantly. My informed guess is that there simply isn't any process that works with hundreds of people that isn't a bit laborious to operate within. There's also no way to flawlessly balance the desire for objective, consistent outcomes with the desire to recognize exceptional individuals.

There's a lot of room for improvement in these processes, and they can absolutely always be improved, but the tension is inherent to the participants' conflicting goals. These conflicting goals are real, fundamental, unavoidable, and must be kept in mind as you make decisions about how your process works.

Performance and Promotions

I'll start by talking about performance processes, including promotions.

FEEDBACK SOURCES

The baseline performance process requires each manager to provide written feedback for each of their direct reports, including a decision on whether to promote them, but there are quite a few details and variations to consider.

The first variations to consider are whether to include peer and upward feedback (on each individual's manager). Because employees almost always have exactly one manager, asking for upward feedback doesn't create too much work. In the worst case, asking for upward feedback is of little value, because individuals don't want to criticize their manager, but it doesn't take up too much time.

Peer feedback can take up a significant amount of time, particularly for highly connected folks who may be asked to provide peer feedback on 10 or more individuals. This is usually accompanied with the advice that you can decline peer feedback requests if you get too many, but many individuals find it difficult to decline peer feedback requests, even if they know they should.

More importantly, my experience is that peer feedback is very inconsistent. I've managed teams that feel peer feedback is too uncomfortable to give honestly, and those teams have provided useless peer feedback: in those cases, it's not worth collecting peer feedback. I've also managed teams that believed feverishly

in the value of peer feedback, and those teams generated insightful, valuable feedback. As such, I prefer to lean toward empowering managers to make the decision on collecting peer feedback for their team. Often, this is a policy decision enacted for the overall company, and in that case it's not a battle I'd pick.

TITLES, LEVELS, AND LEVELING RUBRICS

Agreeing on performance ratings and who should be promoted is nearly impossible without written criteria that describe the dimensions of expected performance for each level. However, before we can talk about leveling rubrics, first we have to talk about levels.

Most companies have paired titles and levels, such as:

- Entry-level Engineer (Level 3)
- Software Engineer (Level 4)
- Senior Software Engineer (Level 5)
- Staff Software Engineer (Level 6)

The specific levels vary widely across companies (some sites have emerged that show how levels differ across companies), and what is a "Level 3" at some companies might be a "60" at another, and a "601" at yet another. It's fairly common for Software Engineer levels to start at "Level 3," as companies use levels across many functions, and often reserve "Level 1" for entry-level roles or roles with fewer entry requirements.

Titles vary even more widely across the industry and there certainly isn't a universal standard to adopt. If you are in the position of setting titles for your company, I recommend using the fairly typical progression of Entry-level Software Engineer, Software Engineer, Senior Software Engineer, Staff Software Engineering, and Senior Staff Software Engineer. If you're tempted to experiment with new titles, note that the downside is that it makes your hiring process more complex since you have to explain what the titles mean and you will lose some candidates who are worried the non-standard titles will harm their career trajectory.

Once you establish Engineering's titles and levels, the next step is documenting the leveling rubrics that describe expectations for operating within each level (again, there are a variety of sites that collect publicly available leveling rubrics from many companies). This can be a very sizable endeavor and I'd recommend skipping the hardest part by picking a reasonably good rubric that's available

online, creating a working group to tweak the details, and then refining it after every performance cycle to address issues that come up.

Additionally, I'd emphasize a few things that I've learned the hard way over time:

Prefer concise leveling rubrics over comprehensive ones.

There's a strong desire for leveling rubrics to represent the complete, clear criteria for being promoted. The challenge, of course, is that many folks are exceptionally good at gaming specific criteria. For example, Stripe's promotion criteria included mentorship, and I encountered folks who claimed to mentor others because they scheduled a meeting with that person, unrequested, and said that constituted mentorship.

Concise rubrics require more nuanced interpretation, but attempts to game rubrics mean that all options in practice require significant interpretation. You can respond to each attempt at gaming with even more comprehensive documentation, but your rubrics will quickly become confusing to use, more focused on preventing bad behavior than providing clear guidance for the well-intentioned.

Prefer broad job families over narrow job families.

A classic executive decision is whether Site Reliability Engineers and Software Engineers should have different leveling criteria. Let's say you decide that, yes, separate criteria would be more fair. Great! Shouldn't you also have separate rubrics for Data Engineers, Data Scientists, Frontend Engineers, and Quality Assurance Engineers?

Yes, each of those functions would be better served by having its own rubric, but maintaining rubrics is expensive, and tuning rubrics requires using them frequently to evaluate many people. Having more rubrics generally means making more poorly tuned promotion decisions, and creating the perception that certain functions have an easier path to promotion. I strongly recommend reusing and consolidating as much as possible, especially when it comes to maintaining custom rubrics for teams with fewer than 10 people. You'll end up exercising bespoke judgment when evaluating performance on narrow specializations, whether you introduce a custom rubric, and it's less expensive to use a shared process.

Capture the how (behavior) in addition to the what (outcomes).

> Some rubrics are extremely focused on demonstrating certain capabilities, but don't have a clear point of view about being culturally aligned on accomplishing those goals. I think that's a miss, because it means you'll promote folks who are capable but accomplish goals in ways that your company doesn't want. Rubrics—and promotions—should provide a clear signal that someone is on the path to success at the company they work in, and that's only possible if you maintain behavioral expectations.

My final topic around levels and leveling rubrics is that you should strive for them to be an honest representation of how things work. Many companies have a stated leveling and promotion criteria—often designed around fairness, transparency and so on—which is supplemented by a significant invisible process underneath that governs how things actually work. Whenever possible, say the awkward part out loud and let your organization engage with what's real. If promotions are constrained by available budget and business need, it's better to acknowledge that than to let the team spend its time inventing an imaginary sea of rules to explain unexpected outcomes.

PROMOTIONS AND CALIBRATION

With leveling criteria, you can now have grounded discussions around which individuals have moved from one level to another. Most companies rely on managers to make a tentative promotion nomination, then rely on a calibration process to ratify that nomination. Calibration is generally a meeting of managers who talk through each person's tentative rating and promotion decision, with the aim of making consistent decisions across the organization.

In an organization with several hundred engineers, a common calibration process looks like:

1. Managers submit their tentative ratings and promotion decisions.
2. Managers in a sub-organization (e.g., Infrastructure Engineering) meet together in a group of 5–8 managers, including the manager responsible for the sub-organization (e.g., the Director of Infrastructure Engineering who the other managers report into), to discuss each of the tentative decisions within their sub-organization.

3. Managers reporting to the Engineering executive meet together with the Engineering executive, and re-review tentative decisions for the entire organization. In practice, this is too many folks to review in detail, so this round typically focuses on promotions, top performers, and bottom performers.
4. The Engineering executive reviews the final decisions with the People team, and then aligns with other executives to maintain some degree of consistency across organizations. They'll also review how the proposed ratings and promotion decisions will impact the current company budget.

This example has three rounds of calibration (sub-organization, organization, and executives), and each round will generally take the involved managers three to five hours to complete. The decisions significantly impact your team's career, and the process is a major time investment.

The more calibrations that I've done, the more I've come to believe that outcomes depend significantly on each manager's comfort level with the process. One way to reduce the impact of managers on their team's ratings is to run calibration practice sessions for new managers and newly joined managers, to give them a trial run at the process before their performance dictates their team's performance outcomes.

Another way is for you, as the functional executive, to have a strong point of view on good calibration hygiene. You *will* encounter managers who filibuster disagreement about their team and you *must* push through that filibuster to get to the correct decisions despite their resistance. You will also find managers who are simply terrible at presenting their team's work in calibration meetings and you should try to limit the impact on their team's ratings. In either case, your biggest contribution in any given calibration cycle is giving feedback to your managers to prepare them to do a better job in the subsequent cycle.

While most companies rely on the same group to calibrate performance ratings and decide on promotions, some companies rely on a separate promotion committee for the latter decision, particularly for senior roles. The advantage of this practice is that you can bring folks with the most context into the decision, such that Staff-plus engineers can arbitrate promotions to Staff-plus levels, rather than relying exclusively on managers to do so. The downside is that it is a heavier process and often generates a gap between feedback delivered by the individual's manager and the decision rendered by the promotion committee, which can make the process feel arbitrary.

DEMOTIONS

The flipside of promotions are demotions, often referred to via the somewhat opaque euphemism, "down leveling." Companies generally avoid talking about this concept and will rarely acknowledge its existence in any formal documentation, but it is a real thing that does indeed happen.

There are three variants to consider:

Demotion with compensation adjustment

For example, someone's level goes from Senior Engineering Manager (L6) to Engineering Manager (L5), and you adjust their compensation to be appropriate for an Engineering Manager (L5). Equity grants are, of course, particularly messy to adjust in this way.

Demotions without compensation adjustment

Someone's level goes from, say, Senior Engineering Manager (L6) to Engineering Manager (L5), but you do not adjust their compensation down to match the new level. This is good for them, but in most compensation systems they will exceed (or be close to exceeding) the pay band for the previous level, which means they will see very limited adjustments going forward.

Title demotion without level adjustment

Someone's title goes from a Senior Engineering Manager to Engineering Manager (L6), while maintaining the same level (L6). This means compensation will keep treating them in the same way, but organizationally they'll be treated as a member of the lower level (e.g., not publicly considered a Senior Engineering Manager, not included in forums for Senior Engineering Managers, and so on).

All of these approaches are a mix of fair or unfair, and come with heavy or light bureaucratic aftereffects to deal with going forward. These bureaucratic challenges are why most companies try to avoid demotions entirely. Further, the concept of "constructive dismissal" means that demotions need the same degree of documentation as dismissals. It's certainly not a time-saving approach.

I avoided demotions entirely for a long time, but I have found demotions to be effective in some cases. First, there are scenarios where you mis-level a new hire. They might come in as a Staff Engineer (L6) but operate as a Senior Engineer (L5). In that scenario, your options are either to undermine your leveling for everyone by retaining an underperforming Staff Engineer—which will

make every promotion discussion more challenging going forward—or to adjust their level down. I've done relatively few demotions, but few is not zero. I have demoted folks in my organizations, as well as those I directly managed, and the outcomes were better than I expected in every case where outright dismissal felt like the wrong solution.

FLOOR FOR FEEDBACK

When you're designing processes, it's helpful to think about whether you're trying to raise the floor of expected outcomes ("worst case, you get decent feedback once a year") or trying to raise the ceiling ("best case, you get life-changing feedback"). Very few processes successfully do both, and both performance processes focus on raising the floor of delivered feedback.

For example, many companies have a small number of Quality Assurance testers who are creating significant value for the business, but a calibration process focused on consistency in a larger Software Engineering organization will often make it difficult to promote those testers or award them strong performance scores. Alternatively, Uber's initial compensation process allowed total managerial discretion, which did support rewarding outstanding individuals, but had no system for dealing with bias.

Because performance processes usually optimize for everyone receiving *some* feedback, it's unwise to rely on them as the mechanism for giving feedback to your team. Instead, you should give feedback in real time, on an ongoing basis, without relying much on the performance process to help. If you're giving good feedback, it simply won't help much.

This is particularly true as your team gets more senior. If senior folks are only getting performance feedback during the performance process, then something is going very wrong. They should be getting it much more frequently.

Managing New-To-You Functions

One of the trickiest aspects of performance management is when you end up managing a function that you've never personally worked in. You may be well-calibrated on managing a Software Engineer's performance, but feel entirely unprepared to grade Data Scientists or Quality Assurance Engineers. That's tricky when you end up managing all three.

What I've found effective:

- Leave behind your functional biases (e.g., "QA is easy") that you may have developed earlier in your career.
- Don't be afraid to lead, even if you don't know the area well. You are the functional area's executive and if you don't push on performance within the function, no one else will.
- Learn the area's fundamentals: watch them in their workflows, read the foundational texts, attend the tech talks, speak to domain experts in and outside of your company, and so on.
- Find high-judgment individuals internally to lean on, validate ideas with, and consult for input. Be careful how you pick those individuals, as it can go wrong if you lean on individuals that the team doesn't respect.
- Prioritize hiring a functional leader who can operate as the area's quasi-executive. Ultimately, you will never have enough time to become an expert in each area you work in, and that problem will only compound as you move into more senior roles at larger companies.

  This certainly *is* tricky, but don't convince yourself that it can't be done. Most executives in moderately large companies are responsible for functions that they've never worked in directly.

Compensation

As an Engineering executive, you will generally be the consumer of a compensation process designed by your People team. In that case, your interaction may come down to reviewing the proposed changes, inspecting for odd decisions, collecting feedback from senior managers about the proposals for their team, and making spot changes to account for atypical circumstances.

That said, I have found it useful to have a bit more context on how these processes typically work, so I will walk through some key aspects to keep in mind:

Companies typically build compensation bands by looking at aggregated data acquired from compensation benchmarking companies.

Many providers of this data rely on companies submitting their data and try to build a reliable data set despite each company relying on their own inconsistent leveling rubrics. You'll often be pushed to accept compensation data as objective truth but recognize that the underlying data set is far from perfect, which means compensation decisions based on that data set will be imperfect as well.

Compensation benchmarking is always done against a self-defined peer group.

For example, you might say you're looking to benchmark against Series A companies headquartered in Silicon Valley, or Series B companies headquartered outside of "Tier 1 markets" ("Tier 1" being, of course, also an ambiguous term). You can accomplish most compensation goals by shifting your peer group: If you want higher compensation, pick a more competitive peer group, if you want lower compensation, do the opposite. Picking peers is more an art than a science, but it's another detail to pay attention to if you're getting numbers that feel odd.

Once you have benchmarks, you'll generally discuss compensation using the compa-ratio, which expands to "comparative ratio."

Someone whose salary is 90% of the benchmark for their level has a 0.9 compa-ratio, and someone who has 110% of the benchmark for their level has a 1.1 compa-ratio.

Each company will have a number of compensation policies that are described using compa-ratios. For example, most companies have a target compa-ratio for new hires of approximately 0.95 compa and aim for newly promoted individuals to reach approximately 0.9 compa at their new level after their promotion. Another common example is for companies to have a maximum compensation of 1.1 compa-ratio for a given level: after reaching that ratio, your compensation would only increase as the market shifts the bands or if you were promoted.

Every company has a geographical adjustment component of their compensation bands.

A simple, somewhat common implementation in the U.S. is to have three tiers of regions—Tier 1, Tier 2 and Tier 3—with Tier 2 taking a 10% compensation reduction, and Tier 3 taking a 20% reduction. Tier 1 might be Silicon Valley and New York, Tier 2 might be Seattle and Boston, and Tier 3 might be everywhere else. Of course, some companies go far, far deeper

into both of these topics as well, but structurally it will be something along these lines.

Whatever the compensation process determines as the correct outcome, that output will have to be checked against the actual company budget. If the two numbers don't align, then it's almost always the compensation process that adjusts to meet the budget. Keep this in mind as you get deep into optimizing compensation results: no amount of tweaking will matter if the budget isn't there to support it.

Whatever the actual numbers end up being, remember that framing the numbers matters at least as much as the numbers themselves. A team that is used to 5–7% year-over-year increases will be very upset by a 3% increase, even if the market data shows that compensation bands went down that year. If you explain the details behind how numbers are calculated, you can give your team a framework to use to understand the numbers, which will help them come to terms with any surprises that you have to deliver.

How Often Should You Run Cycles?

Everyone has strong opinions about the frequency of their company's performance cycles. If you run them once a year, folks will be frustrated that a new hire joining just after the cycle might not get any formal feedback for their first year. If you run every quarter, the team will be upset about spending so much time on the process, even if the process is lightweight. This universal angst is liberating because it means there's no choice that will make folks happy, so you can do what you think will be most effective.

For most companies, I recommend a twice-annual process. Some companies do performance twice a year, but only do promotions and compensation once a year, which reduces the overall time to orchestrate the process. There's little evidence that doing more frequent performance reviews is worthwhile.

The only place I'll take a particularly firm stand is against processes that anchor to each employee's start date and subsequent anniversaries. For example, each employee gets a performance review on their anniversary of joining the company. This sort of process is very messy to orchestrate, makes it difficult to make process changes, and prevents inspecting an organization's overall distributions of ratings, promotions, or compensation. It's an aesthetically pleasing process design, but it simply doesn't work.

Avoid Pursuing Perfection

In my earlier discussion of hiring processes in Chapter 20, my advice is to pursue an effective rather than perfect hiring process, and that advice applies here as well. There is always another step to take to improve your performance or compensation process' completeness, but good processes keep in mind the cost of implementing each additional step. Many companies with 20 employees provide too little feedback, but almost all companies with 1,000 employees spend most of their time on artifacts of performance. That time could be devoted instead to giving better feedback or spent on the business' underlying work itself rather than creating meta-commentary about that work.

As an executive, you are likely the only person positioned to make the trade-off between useful and perfect, and I encourage you to take this obligation seriously. If you abscond this responsibility, you will incrementally turn middle management into a bureaucratic paper-pushing function rather than a vibrant hub that translates corporate strategy into effective tactics. Each incremental change may be small enough, but in aggregate, they'll have a significant impact.

If you want to get a quick check, just ask your team—particularly the manager of managers—how they feel about the current process and you'll get a sense of whether the process is serving them effectively. If they all describe it as slow and painful, especially those who've seen processes at multiple companies, then it's worth considering whether you've landed in the wrong place.

Summary

This chapter has covered the core challenges you'll encounter when operating and evolving the performance and compensation review processes for your Engineering organization. With this background, you'll be ready to resolve the first batch of challenges you're likely to encounter, but remember that these are extremely deep topics, about which there is much disagreement, and many best practices of a decade ago are considered bad practice today.

Find links to further reading and resources on *https://lethain.com/eeprimer-refs-22*.

Using Cultural Survey Data

Most scaled companies run a periodic survey that asks questions about the experience of working in the company, collaborating across teams, compensation, and so on. The results are tallied together, reports are generated for each manager about their team, and executives get a report about the company overall. These are often called cultural surveys. There are a variety of vendors willing to sell you a solution like CultureAmp or Lattice, and occasionally you'll see companies build their own tool in-house.

Relative to most other running-the-business responsibilities, senior leaders often don't spend too much time on cultural survey results, but they're a great lens into their data literacy and how they prioritize limited resources to address a very broad problem space. This is just as true of your existing executive team—and even you—as it is of potential hires, and they're a valuable mechanism for earning credibility with your team by following through to address their concerns.

In this chapter, we'll work through:

- Reading survey results
- Taking action on survey data
- Whether to modify survey questions
- When to start and how frequently to run

Even if you haven't run a cultural survey before, by the end of this chapter you'll have a good grasp on how to take advantage of this sort of data.

Reading Results

My quick advice for effectively reviewing survey data is:

- Spend a couple hours digesting the results.
- Focus more on absolute scores than on relative scores (a high score is still high, even if it's relatively down a bit).
- Take the time to identify all issues, rather than getting caught up in the first couple you notice.

You can do reasonably well by following those three steps, but as I've reviewed more and more of these surveys, and managed teams that reviewed results of their own, I've formalized the details a bit more. The approach that I follow today is:

1. Before starting, check if you have whole company access or only have access to the Engineering report. If it's the latter, bring it up at the next executive team meeting. If your executive team leads the company but isn't trusted to view the whole company's survey results, then something doesn't make sense there and it should be fixed. The executive team may find it uncomfortable to see each other's survey results, but that transparency to one another is part of an executive team gelling together. Further, it's very helpful to see if two functions (e.g., Engineering and Product) in a given business unit are both frustrated, rather than having to reverse engineer that narrative by hand. The broader comments are also helpful, as you may identify areas where other functions are frustrated working with Engineering, which is valuable feedback that's easily lost without full report access.
2. Start by creating a private document to collect your notes on the survey. For each major theme you identify, take a screen capture of the data that raises the issue, and add a few sentences of commentary. This is your staging ground for analysis, not your final product, so don't worry about keeping it tidy.
3. Get a sense of the size of your populations in the report. This helps to build an intuition around what kinds of results might be statistically significant. Most changes will be significant in an Engineering organization of 2,000, but if your technical program manager team is only seven people, then even an extreme change probably isn't significant. Survey

tools are strongly incentivized to produce narratives, even if the narrative is relatively weak. This means that you have to be cautiously skeptical of the analysis, particularly any that lacks statistical significance. That's not to say that you should ignore anything without statistical significance, but instead use the right sort of data: comments, follow-ups, and so on, rather than raw numbers.

4. Skim through the entire report and start to group insights into three categories: things to celebrate, things to proactively address, and things to acknowledge. The groupings aren't final, so don't spend too much time on them. Make sure to capture highlights in addition to concerns: especially when things are going poorly, you'll never want to present a purely negative update.
5. If you identified areas of investment after your last survey, how did you perform there? If you saw an improvement, highlight the success. If you didn't, form a hypothesis for why you didn't have an impact.
6. Focus on your highest and lowest absolute ratings. Have these changed? Did you expect them to change?
7. Focus on ratings that are changing the fastest. What's driving the rapid shift?
8. Identify what stands out when you compare across cohorts. How do different managers perform? How about cohorts of tenure, gender, and location?
9. Read every single comment and copy the most relevant ones over to your notes document. What are the subjective pieces of feedback that change how you interpret the data? What are comments you might quote when sharing a summary?
10. End this analysis process by spending an hour with a peer (e.g., the head of Product) to talk through your findings. Do this after they've spent time with their own report. Sometimes you'll identify connected themes across the organizations, but more importantly you'll often find gaps in your analysis by talking it through. You can, in theory, do this with someone on your team, but you'll end up either putting them in a messy position discussing their peers or having to avoid the thorniest issues, both of which are bad outcomes.

At this point, you will have a good sense of the data, and it's time to move on to the fun part: figuring out what to do with it.

Interviewing About Cultural Surveys

When I was at Stripe, I reworked the hiring process for Director-plus Engineering managers. My goal was to better evaluate the polished senior leaders who always said the right thing. I wanted to find the *real* beliefs and behaviors underneath all the polish. One interview focused on a direct report sharing a mediocre strategy proposal for review, and getting a signal on whether the candidate could give useful feedback on improving the document. I knew that interview worked when a candidate said that they wouldn't even give feedback because the proposal's quality was below their bar to even discuss. Another interview I created focused on how candidates reviewed cultural survey results.

The cultural survey results interview generated some of the most useful signals I've gotten from senior leadership candidates. For example, it gave a clear signal on data literacy (e.g., not orienting on statistically insignificant differences without acknowledging it explicitly), comfort building a mental model on the fly from new data (e.g., so far from the data, it seems like the biggest issues are cross-functional coordination and confusion around what work matters most), and the ability to use questions to assess which issues were worth digging into (e.g., this team seems quite frustrated with their manager and the company's overall executive team; did they have any manager transitions recently or is this a long-running issue?). Even more importantly, it made it clear which leaders felt wholly responsible for their organization versus which viewed their role principally as implementing the broader company's policies.

As you think about designing interviews for executives, look for ways to get them to demonstrate their actual skills performing the real work they'll be doing at your company. In my experience, analyzing cultural surveys falls cleanly into that bucket.

Taking Action on the Results

You'll often see teams spending a great deal of time running the survey, even more time analyzing the results, but never actually using that analysis for anything. This frustrates the team members, who become trained to ignore future surveys. On the other extreme, very earnest leaders commit to fixing everything. Your team will initially love you for this, but it won't hit quite the same way six months later when you have to report that you've made very little progress. Success is walking the middle path: identify one or two important areas where you believe you can make real progress, and then actually do the work.

You probably won't be surprised that I've developed a standard pattern around taking action, as well:

1. If you identify any acutely serious issues (e.g., a manager whose team is in open revolt, or an ethical issue is raised), take action immediately.
2. Use your analysis notes to select two to three areas you want to invest in until the next survey is run (e.g., increasing representation for Staff-plus engineers in key decision-making forums, or increasing the sense of belonging for remote employees).
3. Edit your notes and new investment areas into a document you're comfortable sharing with the broader organization. I would err on being transparent, but a very optimistic sort of transparency: "Most things that are relatively down are still absolutely high, a recent trend down is important but we're still significantly higher than historical averages," and so on.
4. Review this document with your direct reports, likely in your team meeting. Review it with your People or Human Resources partner. Review it with your peers on the executive team. Review the document with two or three trusted individual contributors within Engineering who will likely be more sensitive to the kinds of issues that are easier for you to miss as an executive. As you get feedback from reviewers, think through the feedback and incorporate it using your best judgment. Sometimes reviewers manufacture feedback to have something to say, but my reviewers have always found something important that I've missed, so I think it's important to listen carefully to the feedback here, even if it doesn't initially resonate.

5. For the areas you want to invest in, make sure you have explicit, verifiable actions to take. If these are too abstract, then it will be hard to build your team's trust in you because it will be ambiguous whether you actually followed through on your commitments. For example, measure leadership's cross-office trips rather than merely encouraging cross-office collaboration.
6. Share the document with your organization (via email, chat, or whatnot) and schedule a meeting to discuss the findings and priorities with the entire Engineering organization. I often repurpose the Engineering Q&A sessions (see Chapter 10) for this discussion.
7. Follow up on a monthly cadence on progress against your action items. I often include the updates in my weekly updates to the organization (see Chapter 11).
8. Make sure to mention these improvements when you introduce the next round of cultural surveying. The team will take the survey much more seriously if they know you took action last time. From a mechanical perspective, this will also help you score higher on survey questions related to whether you actually made improvements from last time. Unless you do this, any new hire simply won't know if you've made any improvements.

Although this is a straightforward process, it'll build the team's trust in you and strengthen team culture. Executives are rarely held accountable unless they expose themselves to accountability and this is a valuable opportunity to hold yourself accountable to your team. As you do so, it'll become increasingly easy to hold your team accountable as well.

When to Change the Questions

Some executive teams will debate adding questions every time a cultural survey is about to run. I'm sympathetic to the fundamental concern: questions may be poorly worded and they'll never ask about whatever topic is particularly top of mind. While it's important to ensure you get good coverage in your questions, I generally think the default questions are good enough, and that it's not worthwhile to change.

To explain why, think about the two types of data you'll get from cultural surveys:

- Ratings on various questions, which are valuable because you can compare across cohorts and they become even more useful as you can see their trend over time
- Free-form responses that provide subjective context

The first sort of question is expensive to change because each change wipes out any historical context on the question. Changes away from the default questions also mean you can't compare your results against wider industry benchmarks. I would generally recommend against changing the first type of question unless folks taking the survey are frequently confused by a particular question. For example, you'll often hear confusion between terms like "the Management Team" and "your manager" because the wider company doesn't know that the executive team refers to itself as "the Management Team." This kind of confusion has already invalidated the results, so a change that increases clarity is still more valuable than preserving continuity.

The second sort of question can be changed without impacting historical data because there is no history to wipe out. However, my experience is that if folks have something to say, they will say it anywhere, even if the question is somewhat unrelated. Adding more free-form questions doesn't change that, so as long as you have a handful of open-ended questions, I wouldn't worry about adding more.

Taken in sum, it's worth evaluating these questions before your first rollout, but repeated arguments about changing these questions is just executive bikeshedding.

Starting and Frequency

In small organizations, where you have a genuine relationship with every member of the team, you probably won't get too much from running a cultural survey. However, as your organization grows, they become more and more useful. I find Dunbar's number, roughly 150 people, to be a good point to ensure you're running one of these surveys.

Most organizations run these twice a year, which is a reasonable amount. If you want a more principled way to determine the right frequency, consider how much time you're willing to invest in addressing the issues that get raised. If you're already struggling to address concerns from biannual surveys, then

it doesn't make sense to run more frequent surveys. You'll just annoy folks who take them, who will complain that you're running another survey before addressing the previously raised concerns.

Summary

In this chapter, you've dug into using cultural survey data to help operate your organization. So many leaders avoid engaging with this data because it's uncomfortable to get rated, but there are few more valuable sources of data to become a better leader for your organization.

While I've often been proud of the cultural survey scores I receive, sometimes I've been pretty embarrassed. I've been called out for not paying attention to the team's priorities. I've tried to prioritize the remote-working experience, but had it regress rather than improve. I've had below-average scores and bad scores. If you receive this type of feedback, take a moment and be uncomfortable with it. It means that you care! Then, lean into the feedback and use it to motivate you to address the issues at hand. It's normal to feel somewhat uncomfortable, and maybe even a bit defensive, when working through this sort of feedback.

Find links to further reading and resources on *https://lethain.com/eeprimer-refs-23*.

Leaving the Job

If two friendly executives meet for dinner, it's likely they start by exchanging just how messed up things are at work. Initiatives are behind, layoffs are happening everywhere, the team is in disarray. Then they'll laugh and switch topics. Sometimes one of the executives can't navigate the switch and will keep ranting throughout their meal. Having problems is part of being an executive, but when you're that second executive who can't turn off the frustration, it's time to start thinking about leaving.

Departing an executive job is much messier than leaving an individual contributor role and will significantly impact the team and company around you. It's also potentially impactful to your resume; I frequently talk with executives who hate their job but don't want to leave because they're worried it looks bad. "If I just make it to two years, it'll look great."

The optics are real and I understand why people get caught up in them. I've certainly gotten caught up in my own optics at times. That said, the deeper I get into my own career, the more convinced I become that we should think about departing roles in the context of managing our own energy.

In this chapter, we'll walk through:

- Succession planning before a transition
- Making the decision to leave
- How to think about short executive stints
- Whether to line up another role before leaving
- Telling the CEO
- Negotiating the exit package
- Transitioning out and actually leaving

Finally, we'll end with a discussion about handling the hardest aspect of all: indecision. By the end, you'll have a framework for making the decision to leave and a structure to coordinate your departure.

Succession Planning Before a Transition

In a literal sense, deciding to leave your job is the first step in departing. However, in a smooth departure, the first step happens years earlier: building the team to support your eventual transition out of the business. Even if you intend to remain at your current company forever, life comes at you fast. Certainly, I never imagined I would have a stroke in my mid-thirties, with an infant at home, but then I did. I was fortunate to lead a collaborative team that worked together well in my absence, but that wasn't an accident—it was the byproduct of ongoing succession planning.

This planning doesn't need to be anything heavy. It can look like this:

- In performance reviews, think about what your direct reports are missing to thrive in your role and give them at least one of those areas to focus on in each review.
- As they make those improvements, talk to the CEO about the growth you're seeing in your team. For the one or two with the best chance of eventually succeeding into your role, facilitate them building a relationship with the CEO.
- Every quarter, run an audit of the recurring meetings you attend. For each, can you delegate it to someone on your team? If it's not something you can delegate, partner with someone on your team to improve on whatever is missing before another individual could be effective in that forum. While it's hard to displace yourself from executive-only or board meetings, almost everything else is possible. For example, I've seen a number of companies where the Engineering leadership team alternates running their weekly team meeting rather than the Engineering executive running it exclusively.
- Go on at least one long vacation each year (aim for two weeks, if you can) and explicitly delegate your roles across your team rather than slowing things down. Avoid chiming in on email and chat. Even if mistakes are made, they'll be learning mistakes.

Ultimately, who replaces you won't be your decision (it's the CEO's, as discussed in Appendix B), but doing a small amount of ongoing prep work will

make an internal transition possible. Without this proactive work, it likely won't be seriously considered, which is a shame: So many companies would do well to consider the talent they already have.

Deciding to Leave

There's a certain romance to abruptly quitting a job. I've seen two people legitimately rage-quit their jobs. They went into a meeting, got upset, and quit immediately. One situation involved an individual who reported to me. They threw a piece of paper at my face, where they'd signed their name beneath a brief message that read "I quit, effectively immediately." In both cases, I unironically appreciated their momentary clarity with the situation. They were done.

There's a lot less romance in quitting an executive job. There's the dream that originally convinced you to join, still recognizable but frayed. There are dozens of little frustrations, miscommunications, and disagreements. There are years of preparation ahead of time to facilitate a smooth transition. There's also a huge reservoir of things that have gone well: people you love working with, teams that you've helped build, and company successes you were part of. For many, being an executive is an important pillar of their identity, and that's hard to walk away from.

When talking to executives grappling with this intersection of identity and frustration, there are four questions I ask them to help with making the decision to leave:

Has your rate of learning significantly decreased?

Even frustrating situations can be extremely valuable to you long term, as long as you're learning. It's when the rate of learning starts to slow that you should get particularly concerned.

Are you consistently de-energized by your work?

It's normal to have bad days. It's normal to have bad weeks. You will even have bad months. However, if things are consistently bad, then it's time to consider change. Don't trust your gut here. Instead, keep a journal of your daily energy level for a quarter. If you can't find a trend of good days, it's worth considering moving on.

Can you authentically close candidates to join your team?

Every executive I've known can flip into selling mode. They'll give a warm, balanced, but optimistic view of why you should join their company. Even if you're upset with your role, you should still be able to turn the sell on.

If it starts to feel dishonest to switch into selling mode, that's a sign that you're losing your ability to effectively perform your role.

Would it be more damaging to leave in six months than today?

Sometimes executives inadvertently create a values oasis, where their organization's values differ significantly from the wider company's values. That oasis will feel comfortable for their team, but their team will acclimatize to the oasis' environment such that it can't operate effectively if the executive leaves. Your goal is to create an organization that's successful beyond your tenure with the company. If you believe your organization is drifting away from the company's culture, and you're unwilling to steer it back into alignment, then you should consider if you're putting your team's careers at risk by remaining with the company.

One question that I *don't* recommend anchoring to is how much money you believe you'll lose by leaving. There are two reasons for that. First, you may be able to negotiate an agreement with your company to minimize your departure's financial consequences. Second, people are extraordinarily bad at determining the value of their own compensation package. Many tech executives are sitting on equity packages potentially worth between zero and tens of millions of dollars, and the actual value is simply not knowable. While this is particularly true at private companies, even public companies see their fortunes shift rapidly from year to year. You can't make robust decisions based on predicting the future.

Finally, I would caution against making ultimatums, taking a last stand on something, or whatever. Being an executive is an ongoing negotiation with the CEO and management team on how the company functions. If you're not willing, or comfortable, to continue negotiating with the current group, it's time to move on. Even if you leave, you've still been part of the organization and are partially responsible for the steps they've made, even the ones you disagree with. Being an executive is, ultimately, a compromising position—where you're working inside to effect change—rather than a role where you can ever wipe your hands clean. It's also true that you don't want to negotiate too hard with people who are extremely likely to be called for backchannel references when you interview for your next role.

Distracting Yourself from the Issues at Hand

In "Work on what matters", I wrote about Hunter Walk's idea of snacking: doing work that is easy to complete but is low impact. Snacking happens everywhere, but it's particularly common for executives who are trying to hang on to a job that's no longer serving them.

The best story of my own snacking behaviors comes from my time at Stripe. I was focused on revamping the Engineering organization's approach to operating reliable software and decided that it might also make sense to start an internal book club. It was, dear reader, not the right time to start a book club. Once you start looking for this behavior, it is everywhere, including on your weekly calendar. Snacking isn't necessarily bad—a certain amount gives you energy to redeploy against more impactful tasks—but you do have to be careful to avoid overindulging.

Beyond snacking, which can be valuable when it helps you manage your energy levels, there is a similar pattern that happens when a business or individual goes through a difficult moment: under pressure, most people retreat to their area of highest-perceived historical impact, even if it isn't very relevant to today's problems. Let's call this reminiscing. When you see your very capable CEO start to micromanage words in a launch email, or your talented CTO start to give style feedback in code reviews, take it for what it's worth: they're reminiscing. If you spend the time to dig deeper, they're almost certainly avoiding something else that is sapping their energy.

Some real examples from my experience:

- "The systems I architected never had significant reliability issues." A senior Engineering leader drives a top-down re-architecture of their company's software, one that is often anchored to the design problems from their last in-the-details technical role rather than current needs (e.g., a Grand Migration). They'd be better served by addressing the cultural or skill gaps culminating in the reliability problems, instead of trying to solve it themselves.

- "We need to take more risks in our work." Founders may feel trapped by the slog of meeting financial projections and want to reorganize company efforts toward increased innovation without connecting the dots to how it will meet the financial projections. Typically, this is a throwback to an earlier company phase that is a poor fit for the current phase.
- "The team I hired was much stronger than this team." After years of absence, a founder starts revamping the performance or hiring process to address a perceived gap in hiring, but without the context of why the process has evolved the way that it has. They'd be better served by holding their managers accountable or empowering their People team.

If you find yourself reminiscing or snacking more frequently, it's likely a sign that something isn't quite working for you in your current role. What's changed recently, and should you be thinking about that change directly rather than distracting yourself?

Am I Changing Jobs Too Often?

I often talk to executives who have decided to leave, except for "one little issue." While I believe that these issues are often deliberate impediments raised to avoid making the decision, there's one that comes up frequently enough to address explicitly: should executives stay in their role to avoid a short stint on their resume?

Because the people making hiring decisions about Engineering executives generally are not engineers, the evaluation tends to rely a fair amount on optics and they will place meaningful value on your previous tenures. The intersection between my experience, peer experiences, and discussions with executive recruiters is roughly:

- If it's less than three months, just delete it from your resume and move on.
- If it's more than two years, you'll be able to find another role as long as some of your previous roles have been three-plus years.

- As long as there's a strong narrative, any duration is long enough. For example, if your company is acquired by a larger company, the narrative is that you weren't excited being in the larger environment.
- If a company reaches out to you, there's no tenure penalty. For example, you've been in a role for six months but then you get an email about another role. Because they reached out to you, the short tenure would be because you're very excited about them, and departing is not a reflection on you. (Admittedly, it's a bad look to do this multiple times in a row.)

If you don't fall into any of those scenarios, there probably will be a moderate penalty on your next search, with potential employers asking, "But will they stay if we hire them?" I certainly agree that this is unfair in some cases, but success comes from navigating reality's many warts. In such cases, you may need to prioritize finding a next role where you can succeed for four to five years, even if it's more of a lateral move than an upward one.

Leave With or Without Your Next Role?

Tied into the question of tenure, the easiest way to remove tenure risk from your search is to get your next role before leaving your current role. You'll always have more leverage negotiating from an existing role, and if your primary goal is advancing your career, then I would recommend finding your next role before departing your current role. Similarly, executive searches tend to run slowly. You need to be comfortable living six-plus months without new income to consider leaving your role without another one lined up.

That said, I personally believe the greatest risk to your executive career is running out of energy, so taking a bit of rest might be more valuable than the short-term numbers suggest. What if a three-month rest gives you the energy to continue this work for another 10 years? Then the obvious decision gets a bit less obvious.

Finally, leaving without a role lined up makes it much easier to support your outgoing transition, which often opens up the topic of exit packages, which I'll discuss shortly.

Telling the CEO

After you've made a decision to depart, your next step is telling your CEO. It's important to go into that conversation with a firm grip on your goals. Many CEOs will try to change your mind, but it's important to hold to your plan. If you are

open to changing your mind, then you should be having a career discussion with the CEO, not a departure discussion.

Trust is the underpinning of your relationship with the CEO and fraying that trust will make whatever else is happening just a bit worse, confirming whatever other factors contributed to your initial decision to move on. There's a natural momentum here: once you start the departure discussion, you're going to leave. Even if the CEO leaves the conversation believing you're going to stay, you will leave. It'll just take a few months longer than expected.

The most important points to land in this discussion are:

Departure timeline
: What are your preferred and inflexible timelines for leaving?

What you intend to do next
: Taking on a new role, taking time to recuperate, or whatnot.

Why you're departing
: Assuming you've been clear about issues up to this point, there usually isn't something new to say. You've already said your piece to your CEO, and they've acted upon it to whatever extent they're willing. As such, I'd think of this as preparing the messaging for the board, the wider management team, and so on.

Recommended transition plan
: How do you recommend the CEO handle this transition? Who, if anyone, internally should step into your role, and so on. Keep in mind that once you start this discussion, you are now making recommendations, rather than leading the organization.

Generally, the CEO isn't going to make any decisions in this meeting—they'll just hear you out, say they're going to think this through, and schedule a follow-up to discuss details after they've been able to figure out how they want to handle your departure.

Negotiating the Exit Package

Depending on the discussion you've had with the CEO, you may have a lot of leverage for negotiating an exit package or you may not have any leverage at all. If you've lined up a new role and are giving two to four weeks' notice, then you're very unlikely to negotiate an exit package beyond what you negotiated in your initial contract (as discussed in Chapter 1).

On the other hand, if you're willing to time your exit to the company's preference and you have a good relationship with the CEO, then there's a great deal of leeway in the negotiations. If you agree on a three-month departure, then you might be able to stay on payroll for a month or two after your last day. Or you might be able to continue vesting and receiving medical coverage, but not salary, until your next relevant vesting cliff. They might be willing to approve a secondary stock transaction, even if they typically don't approve such transactions.

None of this stuff is guaranteed—it depends entirely on how much you're willing to facilitate the exit, how the company values your past contribution, whether they like you, and your CEO's preferences. It's worth discussing, though, particularly if you go in with clarity about the one thing that really matters to you (e.g., approving a secondary stock transaction, staying on payroll).

Establish the Communication Plan

Once you've negotiated an exit package, the next topic for negotiation is the communication plan. This plan will describe the shared language for why you're leaving, as well as the timeline of communication. The most important pieces to nail down are:

- A shared description of why you're leaving (which can range from the awkward standard of "leaving to spend more time with your family" to whatever else is mutually agreeable to both you and your CEO)
- When each party will be informed of your departure (other executives, your team, your organization, and so on)
- Drafts of emails and announcements to be made to larger groups, such as emails to your organization and the whole company

You will always have some temptation to tell key individuals sooner than the plan suggests, or to be a bit more honest about the underlying concerns that caused you to consider leaving, but I highly recommend following the plan and keeping concerns to yourself. When you're departing, the biggest service you can do is to support the organization you're leaving behind, which usually means being positive even if there is a layer of messiness underneath the official communication plan. (If you *really* want to share more, do it a year later when you catch up to chat with friends from your previous employer.)

Transition Out and Actually Leave

While there's a great deal you can do before you decide to leave, once you've communicated your decision to leave, you won't be able to change much at the company anymore. It can be tempting to believe you'll finally get the chance to address a long-standing issue, but you're a lame duck. Even if your idea is a good one, you've lost credibility to propose it because you won't be around to deal with the consequences.

The most common mistake I see outgoing executives make is trying too hard to help. At this point, you're a consultant, not an executive. The best you can do in a transition is get out of the way and support your CEO, your team, and the company in the plan that they choose. Once you give notice, it's no longer your Engineering team and you'll have to navigate the abrupt shift from leader to follower. Sometimes this means implementing plans you don't particularly agree with. Even if you disagree with the plans, saying you disagree will only destabilize the team's transition. Do your best to support them, and then move on.

Sometimes you'll see folks stay very involved after their departure. They might keep 1:1 meetings, serve as an explicit mentor to team members they managed, or attend social work functions. I understand the intent, but I strongly recommend against these sorts of prolonged departures. They are usually motivated by a sense of guilt rather than a clear plan to help, and they can undermine the person who replaces you.

Revisiting the Decision

There are many micro-decisions within the broader choice to leave a job and one place I see folks get stuck is revisiting the decision repeatedly, sometimes daily. They have a particularly bad meeting and then recite, "Yup, I'm out of this place." They tell a few external friends that they're done. Two days later, earnings are up and they get excited again. They're too embarrassed to tell their friends they've changed their mind (likely for the Nth time), so they wave the question off when they meet up next, "Oh that, yeah. Just a bad day."

Some people exist in that cycle indefinitely. Earlier in my career, we hired a new manager who I reported to, and that manager told me—in our very first 1:1 on our first day—that he'd made a mistake, hated the company, and should quit. Most of our 1:1s focused on my manager telling me how stupid the company was, and how he thought he should quit or get the CTO fired. It wasn't a particularly inspiring situation for me.

Getting stuck ruminating on the decision is the least helpful thing you can do. You'll spend a tremendous amount of your energy debating, but generally without reaching a clear answer. If the answer was clear, you wouldn't be stuck going in a circle. If you, for whatever reason, can't make a decision right now, then I highly recommend establishing a future decision date. This could be a literal date, "the end of next month," or after a particular set of circumstances, "once we see the impact of our latest product features."

Once you reach that moment, see if you're in a place where you can make a decision. If so, then make it! If not, pick another date, and wait until then to reconsider. It's not that the decision gets any easier, but it helps you spend less time spinning on a decision that you are clearly not ready to make.

Summary

In this chapter, you've spent time considering the last decision you'll make at most companies: the decision to leave. Although you'll only leave once, your departures will play a significant role in defining your career. You've reviewed the importance of ongoing succession planning to make a clean departure possible and you've spent time on how to make the decision, and how to translate your decision into an effective departure for your team, your CEO, and for you.

Find links to further reading and resources on *https://lethain.com/eeprimer-refs-24*.

Closing

Operating as an Engineering executive, I've learned that there's quite a distance between influencing an Engineering organization's mechanisms and being the person ultimately accountable for the success of those mechanisms. Crucial to my discovery were all the books I read along the way. I sought out every relevant book I could find. If a book spoke about being an Engineering executive, I found it, bought it, and read it.

If you picked up this book in a similar moment, I hope that it untangles some of the knotty questions you may encounter in an executive role. I hope it lets you skip some of the pitfalls that I struggled with and that you find a better path than I did. However, I want to acknowledge that no amount of preparation, not even reading a copy of this book smuggled back through time, would have taught me everything I needed to know to succeed as an Engineering executive.

The most important idea that I will leave you with is that while you should prepare, read, and study, the most valuable way to become an effective Engineering executive is *doing the work*. You will not step into these roles fully formed. Instead, the work will combine with your prior experience to mold you. Don't be too anxious about what you don't know; no one knows all of it when they get started and these roles will teach you—sometimes uncomfortably—what you need to know to thrive within them.

If you get lost, confused, or uncertain along the way, I'm obligated to remind you to come back to the chapters in this book that relate to the topics you're dealing with. But even more importantly, I remind you to anchor yourself in *doing the work*. What needs to be done? What might help you move forward? What did you learn from your attempt? Problems this messy rarely have any one solution, and you will find new solutions as you grow. It took you many years to become an excellent software engineer, and it'll take you just as many to become an excellent executive.

The software industry is in its early days and there's much more left for us to learn than we happen to know thus far. Much of what we know today will end up being subtly wrong or inefficient. Dogma will raise the floor of execution quality, but we'll only push the ceiling by continuously questing to improve ourselves and our industry.

Additional Resources

I want this book to answer every important question you'll have about becoming and operating as an Engineering executive, but that's more ground than any book can realistically cover. Being a good Engineering executive encompasses the entirety of the domains of software engineering and executive management, which themselves are too broad to cover in any one book. This appendix collects a number of resources that I have found useful in my career as an engineer, Engineering manager, and Engineering executive. If you're looking for more, I recommend each of these.

Foundational Reading

- Carse, James P. *Finite and Infinite Games*. New York: Free Press, 2013.
- DeMarco, Tom. *Slack: Getting Past Burnout, Busywork, and the Myth of Total Efficiency*. New York: Crown Currency, 2002.
- DeMarco, Tom, and Timothy Lister. *Peopleware: Productive Projects and Teams*. Boston: Addison-Wesley Professional, 1999.
- Drucker, Peter F. *The Effective Executive: The Definitive Guide to Getting the Right Things Done*. New York: Harper Business, 2006.
- Forsgren, Nicole, Jez Humble, and Gene Kim. *Accelerate: Building and Scaling High Performing Technology Organizations*. Portland: IT Revolution Press, 2018.
- Fournier, Camille. *The Manager's Path: A Guide for Tech Leaders Navigating Growth and Change*. Sebastopol: O'Reilly Media, 2017.
- Gerber, Michael E. *The E-Myth Revisited: Why Most Small Businesses Don't Work and What to Do About It*. New York: Harper Business, 2004.

- Grove, Andrew S. *High Output Management*. New York: Vintage, 1995.
- Hogan, Lara. *Resilient Management*. New York: A Book Apart, 2019.
- Kim, Gene, Kevin Behr, and George Spafford. *The Phoenix Project: A Novel about IT, DevOps, and Helping Your Business Win*. Portland: IT Revolution Press, 2018.
- Kim, Gene. *The Unicorn Project: A Novel about Developers, Digital Disruption, and Thriving in the Age of Data*. Portland: IT Revolution Press, 2019.
- Meadows, Donella H. *Thinking in Systems: A Primer*. Chelsea, VT: Chelsea Green Publishing, 2008.
- Rumelt, Richard. *Good Strategy Bad Strategy: The Difference and Why it Matters*. New York: Crown Currency, 2011.
- Watkins, Michael. *The First 90 Days: Proven Strategies for Getting Up to Speed Faster and Smarter*. Boston: Harvard Business Review Press, 2013.

Building Valuable Things

- Cagan, Marty. *EMPOWERED: Ordinary People, Extraordinary Products*. Hoboken: Wiley, 2020.
- Cagan, Marty. *INSPIRED: How to Create Tech Products Customers Love* Hoboken: Wiley, 2017.
- Christensen, Clayton M. *The Innovator's Dilemma: When New Technologies Cause Great Firms to Fail*. Boston: Harvard Business Review Press, 2016.
- Christensen, Clayton M., and Michael E. Raynor. *The Innovator's Solution: Creating and Sustaining Successful Growth*. Boston: Harvard Business Review Press, 2013.
- Elliott-McCrea, Kellan. "How to plan?". Personal blog. Posted October 8, 2022.
- Elliott-McCrea, Kellan. "Software and its Discontents". Personal blog. Posted January 16, 2023.
- Increment Staff. "What planning is like at...". *Increment*. Posted November 2021.
- Jaspen, Ciera. Chapter 7, "Measuring Engineering Productivity", in *Software Engineering at Google: Lessons Learned from Programming Over Time*,

edited by Titus Winters, Tom Manshreck, and Hyrum Wright. Sebastopol: O'Reilly Media, 2020.

- Noda, Abi, Margaret-Anne Storey, Nicole Forsgren, and Michaela Greiler. "DevEx: What Actually Drives Productivity". *Queue* 21, no. 2 (May 2023).
- Perri, Melissa. *Escaping the Build Trap: How Effective Product Management Creates Value*. Sebastopol: O'Reilly Media, 2018 (my notes here).

Leading Your Team

- Elliott-McCrea, Kellan. "On Sizing Your Engineering Organizations". Personal blog. Posted January 21, 2019.
- Lencioni, Patrick. *The Five Dysfunctions of a Team: A Leadership Fable*. Hoboken: Jossey-Bass, 2009.
- Scott, Susan. *Fierce Conversations: Achieving Success at Work and in Life One Conversation at a Time*. New York: Berkley, 2004.
- Stewart, Kevin. "How to build a startup engineering team". *Increment*. Posted November 2019.
- Wong, Jason. "Building a First Team Mindset". Personal blog. Posted June 25, 2018.

Operating as an Engineering Executive

- Elliott-McCrea, Kellan. "Surviving being senior (tech) management". Personal blog. Posted July 16, 2013.
- Hogan, Lara. "How to Spend Your First 30 Days in a New Senior-Level Role". Personal blog. Posted January 9, 2023.
- Hogan, Lara. "Pay fair". *Increment*. Posted November 2019.
- Kaukver, Ott. "The process: How Twilio scaled its engineering culture". *Increment*. Posted November 2019.

Interviewing, Hiring, and Job Searching

- Casado, Martin. "Hire a VP of Engineering". *Andreessen Horowitz*. Posted May 26, 2017.
- Terry, Phyl. *Never Search Alone: The Job Seeker's Playbook*. New York: Collaborative Gain, 2022.
- Wilson, Fred. "VP Engineering Vs CTO". *AVC*. Posted October 31, 2011.

Running Meetings

- Hogan, Lara. "On Better Meetings". Personal blog. Posted January 15, 2017.
- Lencioni, Patrick. *Death by Meeting: A Leadership Fable About Solving the Most Painful Problem in Business*. Hoboken: Jossey-Bass, 2004.
- Sinofsky, Steven. "Reaching Peak Meeting Efficiency". *Medium*. Posted May 6, 2018.

Running Distributed Offices and Teams

- Buriticá, Juan Pablo and Katie Womersley. "A guide to distributed teams". *Increment*. Posted November 2019.
- Cutler, Kim-Mai. "How To Build and Run a Geographically Distributed Engineering Team". *Medium*. Posted February 15, 2018.
- Miranda, Bruno. "Building a Distributed Engineering Team". Personal blog. Posted November 2018.
- Pragides, Ron. "Six Virtues for Distributed Offices". *Medium*. Posted October 8, 2017.

Find links to further reading and resources on *https://lethain.com/eeprimer-refs-A*.

Interviewing Engineering Executives

In Chapter 1, we covered getting hired as an Engineering executive, and it's perhaps even more important to discuss the opposite question: how should companies interview and evaluate Engineering executives? As an Engineering executive, you may not directly run one of these searches, but you'll likely be asked for advice about how to run them. In rarer cases, you will indeed end up running the process to hire your successor, which this appendix will walk you through.

The key topics I want to explore are:

- Avoiding the unicorn search
- How interviewing executives goes wrong
- Structuring your evaluation process
- Focusing on four areas for evaluating Engineering executives

These topics will prepare you to conduct an Engineering executive search that culminates in hiring a leader who can support your company today and will help you avoid bogging the executive team down in a multi-month process.

Avoiding the Unicorn Search

While most companies struggle to evaluate incoming executives effectively, there are some that are great at evaluating executives but nonetheless fail to *hire* effectively because they want a rare intersection of skills. For example, I once saw an Engineering executive search that wanted someone with experience leading a

large Engineering function, with deep go-to-market and Product experience, deep domain exposure to a narrow infrastructure engineering domain, cultural alignment with consensus-based decision making, and a sufficiently strong motor to accelerate company processes.

There is always *some* candidate who fits any mold you define but hiring them gets very challenging. Identifying and hiring them is even harder once you acknowledge the breadth of the error bars inherent to this process. Worse, you usually can't go back to reactivate a candidate after you've passed on them, so even if you later realize an earlier candidate was excellent, that prospect will have already passed you by. If you run a narrow search for too long, by the time you open the search up, you may have already rejected your best potential candidates.

My biggest advice for avoiding the unicorn search is to get the search's sponsor (generally the CEO for an Engineering executive) to spend time before the formal search talking to seasoned Engineering executives to assess the profile. These don't need to be folks the CEO could hire, and the goal isn't really to hire them; rather it's to listen to their feedback on the profile and what could make the hiring opportunity sufficiently compelling that a qualified candidate would accept it. This will take a few weeks but will save months of time in the long run.

How Interviewing Executives Goes Wrong

Typically, the hiring loop for a software engineering role starts out messy and is slowly refined into an effective hiring process as you make more successful hires. This post-hire calibration process is particularly important for reducing the number of false positives and false negatives in your interviewers' feedback. Executive searches only make one hire, but evaluate for a very broad role, which makes these loops even harder to calibrate.

Making things even more challenging is the fact that there's rarely someone wholly qualified to assess the potential Engineering executives, who are being hired to be the most senior technical leader at the company. But despite that gap, you'll have many folks with strong opinions about who should be hired. Combining these messy incentives and challenges, most companies bounce between two interview formats:

Vibes and backchannel

Hiring is heavily weighted on a small number of discussions and backchannel references that provide input on a candidate's previous work. These processes generate very little direct signal, which means that internal colleagues often don't buy into the hires. Even the candidates themselves

may not feel particularly evaluated either, which may cause them to decline the offer.

Kitchen sink

Hiring incorporates a wide range of internal interviewers. This might include an interview with the CEO, Product, Design, People, Finance, Marketing, and Sales, along with four or five folks from Engineering. Introducing this many interviewers, many of whom will be unpracticed at interviewing for this role and may rely on an entirely informal interview, will generate numerous false negatives, often anchoring evaluation to the perspectives of folks without clear evaluation criteria and limited exposure to the role you're hiring.

My experience is that neither of these methods is particularly effective at evaluating candidates, with the former accepting too many candidates and the latter rejecting candidates randomly. Further, these experiences leave the candidates themselves skeptical of the company's decision making.

Structure for Evaluating Executives

Fortunately, it's straightforward to design a reasonable process that's comparable to most Engineering executive evaluations and avoids some of the common missteps. I recommend starting with:

Recruiter screening

Generally, executive searches are run through an executive recruiter, an executive recruiting firm, or a VC recruiting firm. They should lightly filter for candidates' interest in the role and their plausibility for getting an offer. My experience is that executive recruiters are excellent at this filtering as long as you listen to them!

CEO chat(s)

Make sure the CEO and candidate would potentially work well together, and focus on the candidate's understanding of the opportunity and the core challenges. Don't lean out of the challenges: the best candidates know the challenges exist and will be skeptical if you try to avoid or downplay them. Have as many of these as necessary to build conviction that the candidate is plausible and engaged.

2–3 interviews with executive peers

Have two to three executive peers interview candidates. These interviews should explicitly cover the topics discussed in "Four Areas of Evaluation" on page 299 and should have a documented rubric for assessing candidates. A written rubric will particularly reduce the risk of false negatives and false positives, which unstructured executive interviews often introduce.

A 30-minute presentation with a 30-minute Q&A

A short presentation given by the candidate, focused on their understanding of what they'd need to do in their new role, is an excellent way to assess whether the candidate is listening throughout the process and whether they have the executive acumen to operate within your organization. Avoid the temptation to expand attendees, and instead include the executives and CEO who have already met the candidate.

Introducing more attendees will randomize evaluation rather than improve evaluation. For example, bringing in the Chief Marketing Officer (CMO) for the first time will often cause the CMO to observe that the presentation missed several key marketing needs, which is somewhat expected if the candidate hasn't yet met anyone from Marketing. That's a randomizing signal. If the CMO is indeed a key stakeholder then they should be one of the executive peers included in the previous step, rather than added at the presentation stage.

Rigorous backchannel references

Find three to four individuals who have worked directly with the candidate for an extended period of time. I'm generally ambivalent about backchannels, but in the case of executives I believe the risk of hiring a poor executive is high enough that it's an essential step. Candidate-supplied references are not a very high signal at this level because the candidate will prepare their reference with talking points, including how to answer questions around gaps.

2–3 interviews with members of Engineering

Assuming the other steps have gone well, end with several interviews with engineers and Engineering managers. Your primary goal here is to build commitment to the candidate from within the Engineering team, but you also want to listen for any major concerns from Engineering. Your interviewers should be running a structured session with explicit areas

of evaluation. Although not ideal, it's acceptable to get some lukewarm signals at this stage, as long as you don't get any major concerns. It's very rare to hire any new manager whose team doesn't have some concerns.

At this point in the process, either go to offer or decide not to extend an offer. Resist the temptation to hedge. Delaying sends candidates a bad message, and there's rarely additional information out there that will change your mind for the better.

Four Areas of Evaluation

There are more skills than you can viably assess in executives. I recommend drilling in on these four areas.

EXECUTIVE SKILLS

Are they an effective listener and communicator? Do they have the fundamental skills expected of an executive at this level, such as operating to a financial plan, supporting a single or multi-business unit organization, running a hiring or performance review process, and so on? You'll get a signal on this from the candidate's presentation, sessions with peer executives, and backchannel references.

ROLE AND COMPANY-SPECIFIC SKILLS

Every executive role you're hiring for is aimed at solving a handful of specific problems at your company, and you should assess on those dimensions. In some cases, the goal is improving partnership between Sales and Engineering. In other cases, it's improving Engineering's velocity. And in others, it's partnering more effectively with peer executives. Identify these and ensure that you explicitly cover them in either the CEO or peer executive sessions.

ENGINEERING FUNCTIONAL EXPERTISE

Depending on how you've scoped your Engineering executive role, you're going to want your hire to have some sort of functional expertise. In some companies, this is being able to go deep on running a scaled organization, guiding new product development, and facilitating partnerships among Engineering and commercial functions, technical architecture, or even infrastructure. Whatever expertise you're looking for, you should ensure that either the Engineering interviews or the peer executive interviews cover these points.

HISTORICAL PERFORMANCE AND BEHAVIOR

Use your backchannel references to get an accurate understanding of the candidate's true performance and behavior over time. There are effective executives who leave a trail of angry peers behind them. Similarly, there are very ineffective but beloved executives who remain far too long at companies that they serve poorly. You can assess self-awareness by asking about these directly, but you can only assess actual performance by talking to folks who were there. Good executives can spin even the worst performance into something positive, which means you simply cannot rely on them to self-evaluate.

You'll note that I've not provided a checklist of skills to evaluate against. This is deliberate, because the role of a strong Engineering executive is exceptionally broad and reducing it to a list of skills will distract you from evaluating what is particularly valuable to your search. Most current CTOs and VPs of Engineering out there are the wrong fit for your role at your company. Evaluate on the specifics, not the universal.

Summary

You now know how to avoid the unicorn search, how to avoid bringing too many interviewers into the process, and how to evaluate the particulars of what you need rather than anchoring too heavily to vibes. Even with all of that in mind, these are still difficult searches. They're even more difficult if you're running them for your replacement or in partnership with a CEO who disagrees with you on what really matters. Don't get discouraged if it takes you five or six candidates before you find someone you're excited about; this is a natural part of learning how to hire a new executive role. Conversely, *do* get worried if you're not excited after talking to 10 or more candidates; that probably means your search is going a bit off the rails.

Reading a Profit & Loss Statement

Some years ago, I was explaining to my manager that I was feeling a bit bored, and they told me to learn how to read a Profit & Loss (P&L) statement. At the time, that sounded suspiciously like, "Stop wasting my time," but operating in an executive role has shifted my perspective a bit: this is actually a surprisingly useful thing to learn. The P&L statement is a map of a company's operation and is an effective tool for pointing you toward the most pressing areas to dig into.

While there is a lot of depth to reading a P&L, I'll walk you from zero to one. In this appendix, we'll cover:

- The components of a P&L statement
- The steps to review a P&L
- An example of applying those steps
- Instructions for finding public companies' P&L statements to practice on

You may not be applying to Chief Financial Officer roles after finishing this appendix, but you should be more comfortable in executive meetings where financials are discussed.

What's in a P&L Statement

In order to review a P&L statement, we need a P&L statement to read, and I've selected the "Summary Consolidated Financial Data" section from page 18 of HashiCorp's S-1 filing, shown in Figure C-1. It'll be helpful to have that at hand to refer to throughout.

	Year Ended January 31,			Six Months Ended July 31,	
	2019	2020	2021	2020	2021
	(in thousands, except share and per share data)				
Consolidated Statements of Operations Data:					
Revenue:					
License	$ 5,610	$ 18,503	$ 36,208	$ 15,204	$ 21,958
Support	43,462	96,820	165,607	75,622	110,888
Cloud-hosted services	972	2,339	4,092	1,341	6,342
Total subscription revenue	50,044	117,662	205,907	92,167	139,188
Professional services	3,807	3,599	5,947	2,625	2,837
Total revenue	53,851	121,261	211,854	94,792	142,025
Cost of revenue:					
Cost of license(1)	169	294	536	243	130
Cost of support(1)	7,619	17,704	27,194	13,469	16,684
Cost of cloud-hosted services(1)	156	1,390	4,811	1,164	5,197
Total cost of subscription revenue(1)	7,944	19,388	32,541	14,876	22,011
Cost of professional services(1)	1,449	4,527	8,511	4,340	3,584
Total cost of revenue(1)	9,393	23,915	41,052	19,216	25,595
Gross profit	44,458	97,346	170,802	75,576	116,430
Operating expenses:					
Sales and marketing(1)	39,386	89,308	141,018	75,951	88,869
Research and development(1)	20,612	40,118	65,248	34,314	43,048
General and administrative(1)	32,337	24,137	48,545	32,835	25,028
Total operating expenses	92,335	153,563	254,811	143,100	156,945
Loss from operations	(47,877)	(56,217)	(84,009)	(67,524)	(40,515)
Other income, net	694	3,382	756	543	89
Loss before income taxes	(47,183)	(52,835)	(83,253)	(66,981)	(40,426)
Provision for income taxes	168	535	262	346	61
Net loss	$ (47,351)	$ (53,370)	$ (83,515)	$ (67,327)	$ (40,487)
Net loss per share attributable to common stockholders, basic and diluted(2)	$ (0.87)	$ (0.90)	$ (1.32)	$ (1.09)	$ (0.61)
Weighted-average shares used to compute net loss per share attributable to common stockholders, basic and diluted(2)	54,238,742	59,161,264	63,375,470	61,696,317	66,076,683
Pro forma net loss per share attributable to common stockholders, basic and diluted(3)			$		$
Weighted-average shares used to compute pro forma net loss per share attributable to common stockholders, basic and diluted(3)					

Figure C-1. Summary financial data from HashiCorp's S-1 statement

There's a lot there! First let's look at the top headers. The first three columns are showing revenue for 2019 through 2021. All numbers here are "in thousands," meaning that $18,503 is actually $18,503,000, and so on. The last two columns look at six-month windows. We're trying to get a sense of the business overall, so let's focus on the annual numbers.

When you're looking at internal data, it's particularly important to understand where a table shifts from historical data to forecast data. It's very common when talking with an early-stage startup to see a somewhat optimistic current year forecast as part of their financial data. Public companies, on the other hand, essentially never share forecasts. All S-1s, 10-Ks and similar documents are historical data. There are cases when reported data initially appears to be a forecast (for example, some companies end their financial year in January), which is the case here for GitLab: their 2021 financial year only runs through January 31, 2021, so their 2021 results are not a forecast even though this report was issued on November 4, 2021. (Consequently, the majority of GitLab's 2021 performance would only become available in their 2022 finance year report.)

Now we'll dig into Figure C-2, which is the same data as before, but cut down to only focus on annual data.

	Year Ended January 31,		
	2019	2020	2021
		(in thousands, except share and	
Consolidated Statements of Operations Data:			
Revenue:			
License	$ 5,610	$ 18,503	$ 36,208
Support	43,462	96,820	165,607
Cloud-hosted services	972	2,339	4,092
Total subscription revenue	50,044	117,662	205,907
Professional services	3,807	3,599	5,947
Total revenue	53,851	121,261	211,854
Cost of revenue:			
Cost of license(1)	169	294	536
Cost of support(1)	7,619	17,704	27,194
Cost of cloud-hosted services(1)	156	1,390	4,811
Total cost of subscription revenue(1)	7,944	19,388	32,541
Cost of professional services(1)	1,449	4,527	8,511
Total cost of revenue(1)	9,393	23,915	41,052
Gross profit	44,458	97,346	170,802
Operating expenses:			
Sales and marketing(1)	39,386	89,308	141,018
Research and development(1)	20,612	40,118	65,248
General and administrative(1)	32,337	24,137	48,545
Total operating expenses	92,335	153,563	254,811
Loss from operations	(47,877)	(56,217)	(84,009)
Other income, net	694	3,382	756
Loss before income taxes	(47,183)	(52,835)	(83,253)
Provision for income taxes	168	535	262
Net loss	$ (47,351)	$ (53,370)	$ (83,515)
Net loss per share attributable to common stockholders, basic and diluted(2)	$ (0.87)	$ (0.90)	$ (1.32)
Weighted-average shares used to compute net loss per share attributable to common stockholders, basic and diluted(2)	54,238,742	59,161,264	63,375,470
Pro forma net loss per share attributable to common stockholders, basic and diluted(3)			$
Weighted-average shares used to compute pro forma net loss per share attributable to common stockholders, basic and diluted(3)			

Figure C-2. Financial year data from HashiCorp's S-1

Let's take a look at the rows. The major sections are:

Revenue across a few different business lines (e.g., license, support)

This is how much money is booked by each business line. For example, "Support" brought in $43m of revenue in 2019.

Cost of revenue for each business line

This is the cost of producing the revenue. For example, "Cloud-hosted services" spent $156k in 2019 (to generate their $972k in revenue).

Gross profit

This is "total revenue" minus "total cost of revenue." This is how much profit the company would make if it had no operating expenses. For example, $170m in gross profit in 2021.

Operating expenses

This is how much money was spent operating the business. For example, $153m in 2020.

Loss from operations

This is "gross profit" plus "other income, net" minus "operating expenses." For example, $84m in 2021.

Net loss

This is "gross profit" minus ("loss from operation" plus "provision for income taxes"). For example, $47m in 2019.

There are a few more rows, but everything else you can ignore from the perspective of understanding the business.

Finally, it's worth taking a moment to dig into GAAP versus Non-GAAP. The Financial Accounting Standards Board defines the accounting rules known as GAAP, Generally Accepted Accounting Principles, and most financials you'll see will label themselves as either GAAP-compliant or not (e.g., this segment of HashiCorp's 10-K statement). Figure C-3 shows how GAAP accounting is emphasized in a typical report.

The following table presents our cash flows for the periods presented and a reconciliation of free cash flow and free cash flow margin to net cash provided by (used in) operating activities, the most directly comparable financial measure calculated in accordance with GAAP:

	Year Ended January 31,		
	2022	2021	2020
		(in thousands)	
GAAP net cash used in operating activities	$ (56,215)	$ (39,623)	$ (28,365)
Add: purchases of property and equipment	(214)	(4,304)	(980)
Add: capitalized internal-use software	(6,382)	(2,920)	-
Free cash flow (used in)	$ (62,811)	$ (46,847)	$ (29,345)
GAAP net cash used in operating activities as a percentage of revenue	(18)%	(19)%	(23)%
Free cash flow as a % of revenue	(20)%	(22)%	(24)%

Figure C-3. Document with explicit emphasis around using GAAP accounting

There is little consistency in how companies calculate their non-GAAP financials, which makes them tricky to draw conclusions about. Most frequently, companies exclude non-recurring or one-time expenses. Assuming you're looking at your company's internal P&L, the best bet is to ask someone on your Finance team to explicitly walk you through how any non-GAAP figures differ from the GAAP definition. Often, non-GAAP gives a clearer understanding of a business' operational state, but the goal of any non-GAAP measure is always crafting a narrative. Make sure you understand the motivations behind that narrative!

Learning from a P&L

Now that we've covered the individual components of the P&L, let's dig into actually analyzing it in Figure C-4. The first step to take is to move the P&L into our own spreadsheet so we can do a bit of basic math, in this case using Google Sheets. We'll start with columns tracking year-over-year (YoY) growth.

	2019	2020	YoY %	2021	YoY %
Revenue					
License	5,610	18,503	230%	36,208	96%
Support	43,462	96,820	123%	165,607	71%
Cloud-hosted services	972	2,339	141%	4,092	75%
Total subscription revenue	50,044	117,662	135%	205,907	75%
Professional services	3,807	3,599	-5%	5,947	65%
Total revenue	53,851	121,261	125%	211,854	75%
Cost of revenue					
Cost of license(1)	169	294	74%	536	82%
Cost of support(1)	7,619	17,704	132%	27,194	54%
Cost of cloud-hosted services(1)	156	1,390	791%	4,811	246%
Total cost of subscription revenue(	7,944	19,388	144%	32,541	68%
Cost of professional services(1)	1,449	4,527	212%	8,511	88%
Total cost of revenue(1)	9,393	23,915	155%	41,052	72%
Gross profit	44,458	97,346	119%	170,802	75%
Operating expenses					
Sales and marketing(1)	39,386	89,308	127%	141,018	58%
Research and development(1)	20,612	40,118	95%	65,248	63%
General and administrative(1)	32,337	24,137	-25%	48,545	101%
Total operating expenses	92,335	153,563	66%	254,811	66%
Loss from operations	-47,877	-56,217	17%	-84,009	49%
Other income, net	694	3,382	387%	756	-78%
Loss before income taxes	-47,183	-52,835	12%	-83,253	58%
Provision for income taxes	168	535	218%	262	-51%

Figure C-4. Simplified view of financials showing year-over-year changes

Once you have a version of the table you can edit, explore the statement by taking the following steps:

1. Understand the relationship between revenue and cost of revenue, by business lines.
2. Explore the delta between historical trend and forecast projections (only if you're looking at a private company's P&L, which includes a forecast; this won't apply for public data).
3. Write down what you find to be surprising.
4. Identify further actions to understand those surprising areas.

We'll handle the final step once we've collected our questions. Before that, we'll go through the table row by row and apply the first three steps, starting with understanding revenue:

- License revenue grew by 230% ('19-'20) and then 96% ('20-'21).
- Support revenue grew by 123% ('19-'20) and then 71% ('20-'21).
- Cloud services revenue grew by 141% ('19-'20) and then 75% ('20-'21).
- Total revenue grew by 125% ('19-'20) and then 75% ('20-'21).
- My sense of cloud services is that they're going at an OK clip, but a bit slower than expected given that their absolute size (4m in 2021) is relatively low. Purely from a financial analysis perspective, I'd wonder if maybe cloud services don't yet have a strong product-market fit or perhaps are having some go-to-market challenges.
- In addition to cloud services, professional services is also small and growing slowly.
- Altogether, these are reasonable revenue growth rates. In particular, the support revenue growth is strong given the absolute size of the business.

Another question for each of these business lines is what percentage of revenue is coming from renewing customers and what percentage is coming from new business? The core of a good SaaS business is a healthy renewal rate: you need a much smaller Sales team to drive revenue growth if the product does a good job of retaining existing revenue.

Next, let's look at "cost of revenue." What's really interesting are places where cost growth is accelerating or decelerating relative to revenue growth. For

example, license revenue grew by 96% in '20 to '21 and costs grew a bit slower, at 82% in that same period. This means they're achieving some economies in their growth. Conversely, it will definitely be interesting to dig into cloud services revenue and costs, where revenue only grew 75% from '20 to '21 while costs grew 246%. Although the strategic role of cloud services is unclear, the P&L doesn't tell a particularly optimistic story about its trajectory.

Sometimes the interesting questions come from shifts over time. For example, it's interesting that growth in support costs outpaced support revenue growth in '20, but not in '21. Understanding why support costs grew more slowly in '21 than in '20 will lead to a valuable insight into how that business truly operates.

If we look at total revenue growth versus total cost of revenue growth, there's a slightly concerning trend that revenue is growing slower than costs in '20 (125% in revenue versus 155% in cost of revenue) and just barely faster than costs in '21 (75% in revenue versus 72% in cost of revenue). However, this is a situation where looking at percentage growth is a bit misleading. The absolute values tell a much healthier story, as is made clear in the gross profit row with strong positives across the line.

Next up is operating expenses. The "Sales and marketing" (S&M) spend accelerated a fair bit in '20 and then decelerated in '21 on a relative basis. However, on an absolute basis, S&M costs grew by about $50m both years. That's a large increase and it would be worth digging into where that spend is going. It's often helpful to look at the relative size across operating sections, and I personally find it a bit surprising that "General and administration" (G&A) is a larger operating cost than "Research and development" (R&D) in 2019 and would love to understand that a bit better.

Finally, a quick look at net loss. This is not a profitable business, and the loss is growing. However, losses grew more slowly than revenue growth in '21 and grew much slower than revenue growth in '20. Understanding what caused that swing (at least part of it was G&A costs doubling from '20 to '21) will make the path to profitability much clearer.

Digging into the Questions

All right, let's end by considering how we might dig into each of the surprising points (from the perspective of an internal executive). What I've found most effective is grouping the questions by the team to follow up with, sending them your questions, and then scheduling time to discuss.

Questions to dig into with the Finance team:

1. What is revenue retention by business line? How much revenue is new versus renewing? (This might be a Sales team question; it depends a bit on company and structure.)
2. Why were G&A operating costs higher than R&D operating costs in '19? Why did G&A costs double from '20 to '21?

Questions to ask appropriate business owners:

1. Why is cloud services growing relatively slowly? (Talk to the Product and Sales teams.)
2. Why are cloud services costs growing 3.5x faster than revenue? Do we expect that to no longer be true at some point in the future?
3. Why did growth in support costs grow faster than support revenue growth in '21 but not in '20? What changed?
4. What is the additional $50m S&M spend in both '20 and '21 going toward? How are we measuring efficiency of that spend?

Questions that should be answered by your management team's strategy:

1. What is the path to profitability?
2. What is our strategy around S&M operating costs versus R&D operating costs?

After having these discussions, you will have a vastly clearer understanding of your business' reality. Before running an exercise like this, you may think that you understand your business, but you were relying on other folks' interpretation to fuel that understanding. Now it's your understanding driving your confidence.

This Is an Ongoing Activity

After you dig through the P&L, you might imagine that you're done. That's true in a discrete sense, but really your understanding is merely paused until the next iteration of the P&L becomes available. For example, we reviewed HashiCorp's S-1, which includes their performance through January 2021. In Figure C-5, you can see the subsequent data they shared: results through January 2022 in their 10-K issued in March 2022. On page 77, they have their 2022 results (meaning, specifically, results through January 2022).

	2019	2020	YoY %	2021	YoY %	2022 (from 10-K)	YoY %
Revenue							
License	5,610	18,503	230%	36,208	96%	47,504	31%
Support	43,462	96,820	123%	165,607	71%	247,566	49%
Cloud-hosted services	972	2,339	141%	4,092	75%	18,613	355%
Total subscription revenue	50,044	117,662	135%	205,907	75%	313,683	52%
Professional services	3,807	3,599	-5%	5,947	65%	7,086	19%
Total revenue	53,851	121,261	125%	211,854	75%	320,769	51%
Cost of revenue							
Cost of license(1)	169	294	74%	536	82%	221	-59%
Cost of support(1)	7,619	17,704	132%	27,194	54%	38,080	40%
Cost of cloud-hosted services(1)	156	1,390	791%	4,811	246%	14,031	192%
Total cost of subscription revenue(	7,944	19,388	144%	32,541	68%	52,332	61%
Cost of professional services(1)	1,449	4,527	212%	8,511	88%	11,108	31%
Total cost of revenue(1)	9,393	23,915	155%	41,052	72%	63,440	55%
Gross profit	44,458	97,346	119%	170,802	75%	257,329	51%
Operating expenses							
Sales and marketing(1)	39,386	89,308	127%	141,018	58%	269,504	91%
Research and development(1)	20,612	40,118	95%	65,248	63%	165,031	153%
General and administrative(1)	32,337	24,137	-25%	48,545	101%	112,108	131%
Total operating expenses	92,335	153,563	66%	254,811	66%	546,643	115%
Loss from operations	-47,877	-56,217	17%	-84,009	49%	-289,314	244%
Other income, net	694	3,382	387%	756	-78%	162	-79%
Loss before income taxes	-47,183	-52,835	12%	-83,253	58%	-289,152	247%
Provision for income taxes	168	535	218%	262	-51%	986	276%
Net loss	-47,351	-53,370	13%	-83,515	56%	-290,138	247%

Figure C-5. Updated financial data from 10-K filing

I won't go into the full details, but if you review this spreadsheet combining their S-1 and 10-K results, you can tell they had a challenging year. HashiCorp is far from alone in that regard—almost everyone had a rough year in 2021—but if they rise to this challenge, then they might come out of this adversity as a much more profitable company. This uncertainty is part of why I don't generally recommend folks try to make financially optimal moves during a downturn.

Finding S-1s and 10-Ks

I'll go on a brief tangent on finding P&L statements for public companies. All the numbers shared in this appendix are public record and you can find them by going to Securities and Exchange Commission's EDGAR search and typing in the company's name. Using HashiCorp, for example, I started typing in "hashi," after which search suggested HashiCorp's ticker, "HCP." I then clicked to the

page for HashiCorp, Inc. From there, if you click on "View filings" you can see all the interesting filings, particularly the S-1 and 10-Ks.

Most companies will also have an investor relations (IR) website, like HashiCorp's *ir.hashicorp.com*, with links to their recent filings like this page hosting HashiCorp's quarterly results. Generally, it's easier to use EDGAR, but an IR website will often espouse the company's preferred narrative through their earnings calls and press releases. Even if you do want to understand the company's preferred narrative, I'd recommend reading their P&L without any narrative first to avoid unduly steering your attention in their preferred direction.

Summary

Now that you've worked through this appendix, you should feel confident working through a P&L statement. These documents will never be simple to read because they capture a complex narrative within a few columns, so don't get intimidated if it takes some study to make sense of their trends.

As a closing thought, remember that P&Ls at early-stage companies are often wrong, and they can be wrong in a lot of different ways. Costs can be miscategorized, non-recurring revenue can be booked as recurring revenue, and so on. However, even when a P&L is wrong, it's almost always wrong in an interesting way that will teach you about the underlying business and team running it. If you really want to understand a business starting from scratch, there are few better starting places than their latest P&L statement.

Find links to further reading and resources on *https://lethain.com/eeprimer-refs-C*.

Starting Engineering Hubs

Coincident with the start of the Covid-19 pandemic in 2020, the state of remote and in-person work has been in ceaseless flux. Companies abruptly shifted from mostly working in the office to working from home. This has been followed by an unpredictable return to offices, with both employees and executives trying to find a new equilibrium. Many companies founded in 2023 are leaning heavily into in-office culture. Others are celebrating their commitment to remote work from the very beginning. Holders of both these opposing perspectives are confident they're well-positioned for the future.

With that backdrop, it feels somewhat quaint to write about a small slice of that larger conversation: founding your second Engineering hub, your first Engineering hub outside of your founding office. While this slice is far from the whole topic, my hope is that thinking through this subset will provide some insight into the many office- and location-based challenges confronting a modern Engineering executive.

I've personally found opening hub offices to be particularly instructive because they offer a rare chance to bootstrap an existing company's culture in a new environment. There's been something new to learn each time I've been involved in spinning up an office, from Uber's presence in Lithuania, to Stripe's presence in Seattle, and SocialCode's acquihire of Digg that formed a new San Francisco office.

In this appendix, we'll work through:

- Why I call these "hub offices" rather than "remote offices"
- Why you might consider opening a hub office
- Why offices benefit from a clear office mission

- The executive engagement necessary for a new office's success
- Why new offices are particularly fragile when it comes to unpredictability and an inconsistent approach
- How to integrate a new office into your company's culture

By the end, you'll have a clear sense of the necessary steps to grow a new hub office, and whether this is a commitment you're ready to make.

Hub, Not Remote

Before diving into making hub offices successful, a few words on "remote" offices, which is a term that some headquarters use to refer to their other hubs. It's a poor choice of words: remote offices are far away. Selling folks to join a nascent office depends on them believing it will become an important, central part of your company's success. Every time you use the word "remote" to describe your new office, potential candidates and new hires experience a moment of mental dissonance—how can this be a critical part of the company if leaders keep calling it remote?

The solution is easy: don't call offices remote; reserve the term "remote" for folks who aren't working in any company office.

Why Add an Engineering Hub

Most major company decisions tie back to the company's financial plan, and starting an Engineering hub is no exception. Usually, the discussion around adding a new hub will anchor to one of three observations:

1. Engineering headcount costs are high and would be lower if you hired somewhere with lower costs. This might lead a U.S.-based company to propose hiring in less expensive U.S. markets, or to hire in India, China, Eastern Europe, and South America.
2. Engineering hiring velocity hasn't met targets, and you can hire faster by hiring in multiple markets. You'll particularly hear this from companies with products that are scaling quickly. Such companies are looking to hire engineers who have experience working at significant infrastructure scale.
3. You make a large acquisition of a company with an existing Engineering team that already works in a location where your company doesn't have an existing presence.

Hubs can contribute to solving all of these problems, but they're never a short-term solution. For example, hiring into smaller or cheaper markets will reduce your costs, but you'll usually find less-experienced talent. For those hubs to succeed, you'll need to design around effectively onboarding less-experienced engineers and training them into the sort of talent your organization needs. This is an ongoing, significant investment. If you make the investment, then it will work out, but you can't delude yourself into thinking that merely opening the hub will solve your challenge.

As an executive, you'll often find your board or your CEO is pushing you to make these sorts of changes as miracle, quick fixes. The hard part is forcing them to pick from real alternatives, not cherry-picking the most convenient slices of reality (e.g., the beloved-but-rare low-cost hub where you can hire experienced engineers without making a major investment in training and travel).

Mission

When I joined Uber, the production on-call was so noisy that the on-call primary's phone would sometimes run out of battery charge before their 12-hour shift ended. The long-term solution was shifting from incident recovery to incident prevention combined with reducing spurious alerts. But in the short-term, we decided to grow a full Engineering office in Lithuania to staff a follow-the-sun rotation. (Why Lithuania? It was in the right band of time zones, we already had two fantastic engineers on the team working from Lithuania, and we believed we'd be the premier Engineering employer in the region.)

The team we hired in Lithuania did a phenomenal job of supporting the on-call rotation and as they grew they started tackling more and more of the underlying causes driving the on-call cacophony. As they tackled those issues, they kept running smack into whatever existing team felt they were the owner of the alert-generating system that the Lithuanian team wanted to improve. Change how Puppet was deployed? Another team owns that. Change how service-to-service routing was updated? Another team owns that, too.

After observing this problem a few times, it became clear that we hadn't defined a sufficiently clear and important mission for the hub office. Acknowledging pages is valuable, but it isn't a purpose. It's not a mission. A mission is end-to-end ownership: a business line, a new product, or the entirety of a key piece of infrastructure.

After debugging the conflict, we worked to identify a large, critical area of responsibility that the Lithuania hub office would take full ownership over. We

made sure they could move forward in this area without approval from folks in other time zones, and finally we ensured the mission was important—it had to be something that would cripple the company if it wasn't going well. Something important enough that we couldn't ignore it if it wasn't going well. Defining an important mission didn't fix things immediately, but it created a clear mission for the team, a feedback loop on whether we were sufficiently empowering them to own that mission, and a built-in focus from leadership.

Ensuring each office has an important mission is the single most important thing to do to make an office successful. If you can't identify a clear, ownable mission, then you shouldn't open a new office. If an office's mission isn't important enough for a senior executive to be the mission's sponsor, then it isn't important enough to serve as the office's first mission. As an office grows, it should accumulate multiple missions and some of those will not be as urgent as the first mission—that's all well and good—but the first needs to be critical.

Transitioning a critical mission to an office that isn't well staffed is a tricky procedure, and I've found it works best to split ownership with an existing team and the new office for a short period of time, roughly the first six months. That said, I'd encourage you to err toward moving the mission early, even if the team isn't quite at size yet, rather than waiting to be fully staffed and slowing the new office's culture of ownership.

You'll know the new office has a clear purpose if every member of the company can mention something that office has launched in the past year, and they know at least one thing that they'd go to that office for first.

Executive Engagement

When the acquihired Digg team bootstrapped a new San Francisco office, I was initially the only senior leader operating within the office and felt extremely responsible for the office's success, but we hadn't yet developed the notion of a Site Lead, which meant that I often felt out of position to advocate for the office in certain company processes.

I've come to believe that defining the role of a Site Lead *early* and with explicit responsibilities is the second most essential investment companies need to make in their new offices' success. The first? Having a Site Lead to begin with. The other areas that are particularly important are sufficient initial headcount to create a real sense of community—let's say at least 12 folks or so—and executive attention.

For those 12 folks, commit to having them be on one or at most two teams. Some companies commit significant headcount to a new office—let's add 50 people next year!—but then scatter them so thinly across teams that it ends up either preventing those teams from gelling or keeps ownership of their mission bridged across multiple offices since the teams aren't at sufficient size to own a critical mission.

The simplest rule of thumb for executive attention is that executives need to visit quarterly for the first year or two as part of forging a successful new office. Your physical presence is the clearest emphasis of your priorities, and people in new offices will literally track how frequently you visit.

You'll know your investment is on track if you visit other offices and they still feel like home, and you're mentioning their work at a healthy clip in company-wide events like all-hands meetings.

Predictability

I recently had dinner with a friend who described working at a company that opened a new office, hired a few folks into it, abruptly canceled the new office to instead open a different new office, decided to 10x the second new office over the next year, succeeded in 10x'ing it, planned to 10x it again, and then instead immediately moved into a headcount freeze. Folks hired into the second new office remained surprisingly enthusiastic, but their implementation and hiring pitches grew hesitant—would the second new office succeed and be important in the long run?

Uncertainty is always disruptive, but it's particularly damaging for new offices that lack the history of successfully overcoming uncertainty that a broader company accumulates over time. Without history, there's no resiliency. This makes it very important to maintain a predictable, consistent investment in new offices for their first three-plus years.

I'll give you one escape card here, though, which is that you're allowed a one-year hiring spurt in your new office where you can defy all best practices around team growth. If you want to grow to 50 engineers in your first year, fine. If you want to grow to 10 engineers in your first year and then grow to 100 in your second year, go for it. Just remember you only have *one* exception card and after that you should return to a sustainable hiring rate of approximately 50% year-over-year growth.

You'll know you're staying predictable by measuring your performance against the plan for office growth and ownership.

Integration

When Stripe Engineering went global, we really went global, making major, simultaneous investments in our Engineering offices in both Europe and Asia-Pacific. With a few iterations on the details, the offices were doing quite well, but early on the time zone spread made integrating the offices more challenging than anticipated.

Each company has a series of communication and community touchstones that are its culture's heartbeat, and there are both physical and cultural challenges to inviting more offices to participate in them:

Physical challenges
: If you have enough geographical office spread, then there is no time of day that will be effective for all offices; asking questions remotely feels different than asking them in the same room; video conferencing equipment keeps improving but retains a resolved commitment to inopportune failures.

Cultural challenges
: New offices don't start with years-long relationships with the founders; new offices will be operating in regions that may have a different cultural or language context than the first office and/or the founders; new offices often surprise long-tenured folks by becoming a mirror into how the company's culture has shifted.

There is a standard playbook for ensuring new offices integrate that works surprisingly well:

Send a landing team
: Identify a group of three to five tenured folks from an established office who can move to the new office for the first year, two years, or permanently and become the new office's initial communication and cultural nucleus.

Ensure that senior leaders travel to the offices
: This is more than a drop-in, but each visit should have an office-wide touch point, maybe a Q&A.

Spread opportunity to every office
: Create checks to watch for the creation of implicit two-tier systems, such as all Staff-plus engineers operating in the first office or using a structured DRI selection process.

Rotate time zones for company-wide meetings

Company meetings are important rituals and they happen infrequently enough that they are a powerful signal of your executive team's priorities.

Fully implementing these practices will disrupt the standard operating procedure in your first office and, yeah, that's kind of the point. If you want your new offices to succeed similarly to your existing offices, the burden of accommodation must be spread across all offices rather than concentrated on the most recent ones. If opening a new office doesn't create friction in your existing offices, either you've truly mastered the process or you're probably pushing too much burden on the new office.

You'll know you're integrating your offices well by segmenting the results of your periodic culture surveys by location and ensuring there is little, if any, delta across offices. For a shorter test, ask a few folks in an established office how they've adapted to incorporate the newest office; if they can't think of anything, spend some time reflecting on that.

As an aside, there is another aspect of integration, which is integrating into the city and community that you're building the office in. Having a Site Lead who has long-standing community ties is exceptionally valuable for this, as is doing small-scale community dinners or events. There's not really a playbook on this that I've seen, but it's worthwhile to think about.

Summary

Considering all the ingredients for a successful new office, you might think to yourself that opening a new office isn't worth all of the work. If that's your conclusion, then you probably shouldn't open one! A poorly supported new office is an ignoble thing to be avoided. However, if you are ready to make the investment, then you now know the steps to get an office up and make an impact for your company. If you keep focused on mission, executive engagement, predictability, and integration, the rest will come together as you go.

Find links to further reading and resources on *https://lethain.com/eeprimer-refs-D*.

Magnitudes of Exploration

In Chapter 3, we discussed creating effective Engineering strategies. One challenge with that discussion is that there are very few publicly shared Engineering strategies to learn from. If you don't happen to work at a company with a clearly written strategy, you'll rarely see one. To provide some visibility into what these strategies can look like, the whole of this appendix shows a sanitized version of Stripe's Engineering strategy at a moment in time in 2019.

In short: mostly standardize, exploration should drive order of magnitude improvement, and limit concurrent explorations.

Standardizing technology is a powerful way to create leverage: Improve tooling a bit and every engineer will get more productive. Adopting superior technology is, in the long run, an even more powerful force, with *successes* compounding over time. The trade-offs and timing between standardizing on what works, exploring for superior technology, and supporting adoption of superior technology are the core of Engineering strategy.

An effective approach is to prioritize standardization, and explicitly pursue a bounded number of explorations that are pre-validated to offer *a minimum* of an order of magnitude improvement over the current standard.

Standardization

Standardization is focusing on a handful of technologies and adapting your approaches to fit within those technologies' capabilities and constraints.

Fewer technologies support deeper investment in each, and reduce time spent on cross-technology integration. Narrow standards simplify the process of writing great documentation, running effective training, providing excellent tooling, etc.

The benefits are even more obvious when operating technology at scale in production. Reliability is the byproduct of repeated, focused learning—often in the form of incidents and incident remediation—and running at significant scale requires major investment in tooling, training, preparation, and practice.

Importantly, improvements to an already adopted technology are the only investments an organization can make that don't create organizational risk. Investing into new technologies creates unknowns that the organization must invest into understanding; investing more into the existing standards is at worst neutral. (This ignores opportunity cost; more on that in a moment.)

I am unaware of any successful technology company that doesn't rely heavily on a relatively small set of standardized technologies. The combination of high leverage and low risk provided by standardized is fairly unique.

Exploration

Technology exploration is experimenting with a new approach or technology, with clear criteria for adoption beyond the experiment.

Applying its adoption criteria, an exploration either proceeds to adoption or halts with a trove of data:

1. Proceeds: The new technology is proven out, leading to a migration from the previous approach to the new one.
2. Halts: The new technology does not live up to its promise, and the experiment is fully deprecated.

This definition is deliberately quite constrained in order to focus the discussion on the value of *effective exploration*. It's easy to prove the case that poorly run exploration leads to technical debt and organizational dysfunction—just as easy as proving a similar case for poorly run standardization efforts.

The fundamental value of exploration is that your current tools have brought you close to some local maxima, but you're very far from the global maxima.

Even if you're near the global maxima today, it'll keep moving over time and trending toward it requires ongoing investment.

Each successful exploration slightly accelerates your overall productivity. The first doesn't change things too much, and neither does the second, but as you continue to successfully complete explorations, their technical leverage begins to compound, slowly, subtly becoming the most powerful creator of technical leverage.

Tension

While both standardization and exploration are quite powerful, they are often at odds with each other. You reap few benefits of standardization if you're continually exploring, and rigid standardization freezes out the benefits of exploration.

Employees at Amazon have described the consequences of over-standardization on their internal tools: once-novel tooling struggles to keep up with the broader ecosystem's reinvention and evolution, leading to poor developer experiences, weak tooling, and ample frustration. (I'm certain this doesn't accurately reflect all of Amazon's development experience; a company that large has many different lived experiences depending on team, role, and perspective.)

If over-standardization is an evolutionary dead end, leaving you stranded on top of one local maxima with no means to progress towards the global maxima, a predominance of exploration doesn't work particularly well either.

In the first five years of your software career, you're effectively guaranteed to encounter an engineer or a team who refuses to use the standardized tooling and instead introduce a parallel stack. The initial results are wonderful, but finishing the work and then transitioning into operating that technology in production gets harder. This is often compounded by the folks introducing the technology getting frustrated by having to operate it and looking to hand off the overhead they've introduced to another team. If that fails, they often leave the team or the company, implicitly offloading the overhead.

Balancing these two approaches is essential. Every successful technology company imposes varying degrees of control on introducing new technologies, and every successful technology company has mechanisms for continuing to introduce new ones.

An Order of Magnitude Improvement

I've struggled for some time to articulate the right trade-off here, but in a recent conversation with my coworker Qi Jin, he suggested a rule of thumb that resonates deeply.

Standardization is so powerful that we should default to consolidating on few platforms and invest heavily into their success. However, we should pursue explorations that offer at least one order of magnitude improvement over the existing technology.

This improvement shouldn't improve on a single dimension with meaningful regressions across other aspects, but instead it should be approximately as strong on all dimensions and at least one dimension must show at least one order of magnitude improvement.

A few possible examples, although they'd all require significant evidence to prove the improvement:

1. Moving to a storage engine that is about as fast and as expressive, but is ten times cheaper to operate
2. Changing from a batch compute model like Hadoop to use a streaming computation model like Flink, enabling the move from a daily cadence of data to a real-time cadence (assuming you can keep the operational complexity and costs relatively constant)
3. Moving from engineers developing on their laptops and spending days debugging issues to working on VMs, which they can instantly reprovision from scratch when they encounter an environment problem

The most valuable—and unsurprisingly the hardest—part is quantifying the improvement and agreeing that the baselines haven't degraded. It's quite challenging to compare the perfect vision of something non-existent with the reality of something flawed but real, which is why explorations are essential to pull both into reality for a measured evaluation.

Limit Work-in-Progress

Reflecting on my experience with technology change, I believe that introducing one additional constraint into the "order of magnitude improvement" rule makes it even more useful: maintain a fixed limit on the number of ongoing explorations that can be occurring at any given time.

By constraining the number of explorations we simultaneously pursue—perhaps two or three for a thousand-person Engineering organization—we manage the risk of technical debt accretion, which *protects* our ability to continue making explorations over time. Many companies don't constrain exploration and quickly find themselves in a position where they simply can't risk making any more.

That's a rough state to remain in.

Being able to continue making discoveries is essential, because the true power of exploration is not the first step but the "nth plus one" step, compounding leverage all along the way.

There isn't a single rule or approach that captures the full complexities of every situation, but a solid default approach can take you a surprisingly long way.

I've been in quite a few iterations of "standardize or explore" strategy discussions, and this is the best articulation I've found so far of what works in practice: *mostly standardize, make sure that explorations drive order of magnitude improvement, and limit concurrent explorations*.

Find links to further reading and resources on https://lethain.com/eeprimer-refs-E.

Index

Symbols

A

B

C

D

E

F

G

H

L

M

N

O

P

Q

R

S

T

U

V

W

About the Author

Will Larson is the Chief Technology Officer at Carta and held senior engineering leadership roles at Stripe, Uber, and Calm. He's the author of *An Elegant Puzzle* and *Staff Engineer*, and is a prolific writer on his blog, *Irrational Exuberance*.

Colophon

The cover illustration is original art by Susan Thompson. The cover fonts are Gilroy and Guardian Sans. The text fonts are Minion Pro and Scala Pro; the heading and sidebar font is Benton Sans.