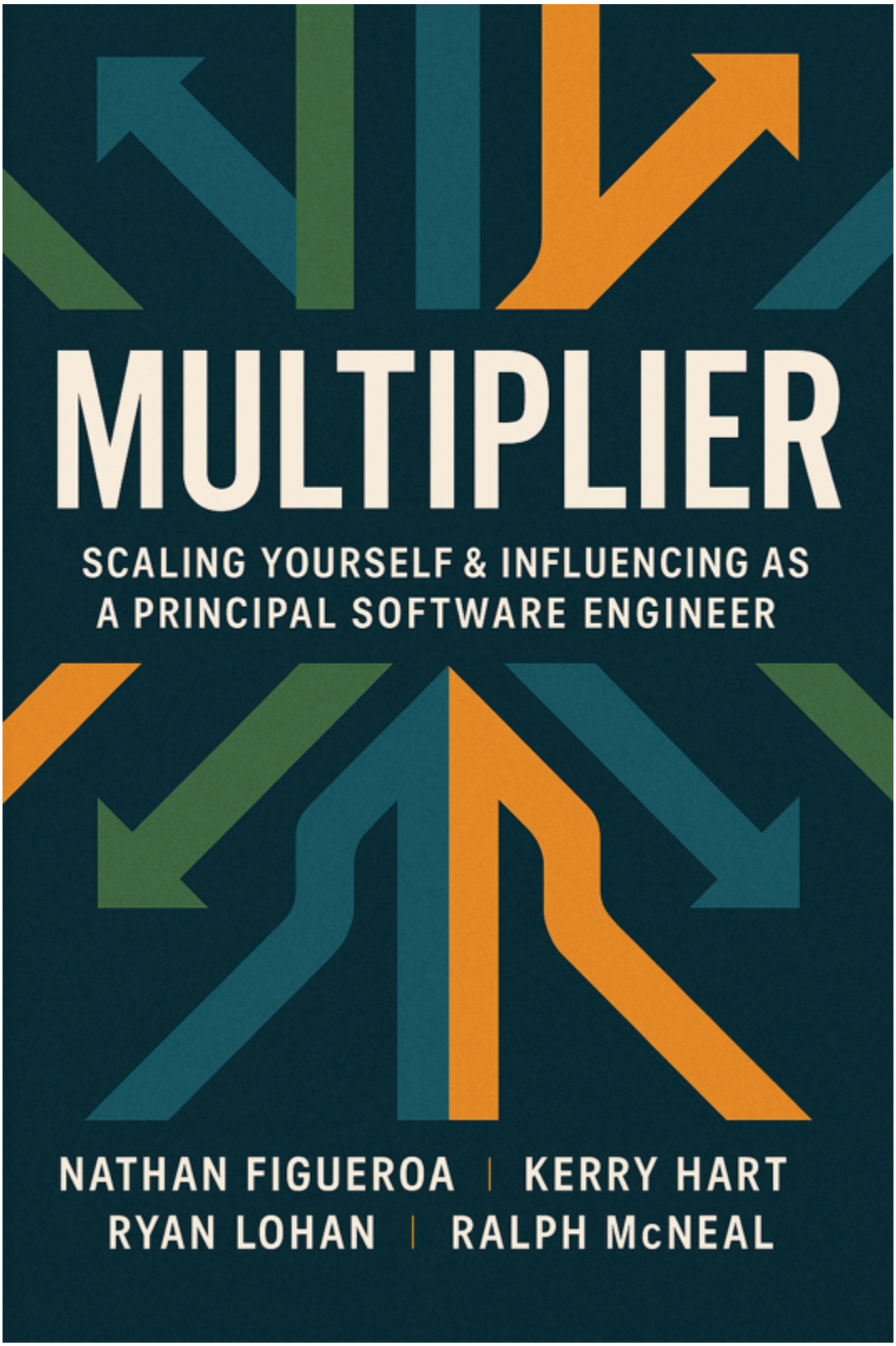

MULTIPLIER
SCALING YOURSELF & INFLUENCING AS A PRINCIPAL SOFTWARE ENGINEER
NATHAN FIGUEROA | KERRY HART
RYAN LOHAN | RALPH McNEAL

# MULTIPLIER

Scaling Yourself & Influencing as a Principal Software Engineer

Nathan Figueroa Kerry Hart

Ryan Lohan Ralph McNeal



Copy edit and page design by Rebecca Moreno.

1st Edition: November 2025

*The practices described in this book reflect experiences within specific organizational contexts. Not all recommendations may apply equally across different organizational cultures or structures. Evaluate and adapt these practices to suit your unique environment. The scenarios and the people in them are fictional. Antipatterns discussed represent common industry scenarios and experiences. Views expressed are those of the authors, not necessarily of Zillow Group.*

# Table of contents

# Why we wrote this

As software engineers grow in our careers, we often must choose among three divergent paths.

The first path stays focused on individual accomplishments: learning to write better code, building better systems, and leveraging new technologies. But the extent of an individual contributor's impact can be limited, especially within large organizations supporting complex products. If we wish to have greater influence, we must take a different path.

The second path is management, where our influence expands through direct leadership. Engineering managers rely on their own technical background to formulate strategy, inspect outputs, assess talent, and otherwise marshal the many engineers they manage to produce the software that underpins the business.

The third path is to become a principal engineer, and it's the path we, the authors of this book, have chosen. The title for this role may vary by company—some refer to it as Staff—but the emphasis on technical leadership remains the same. Our role is deeply technical: we architect systems, write critical code, assess technologies, and otherwise build (and operate) software. Yet, our value is defined not by the consistent delivery of well-defined tasks but by our ability to identify and deliver *outsized impact* (value) for the company. We do this through a mixture of hands-on innovation and strategic influence of others.

For those pursuing the managerial path, there are countless articles, books, and courses — entire curricula! — designed to help us learn and grow. In contrast, there's a dearth of equivalent material for those who wish to grow as principal engineers. We know — we looked!

In our experiences as principal engineers and employees of Zillow Group, we've witnessed firsthand the impact of the lack of structured guidance for this third path. It's difficult to consistently deliver the significant organizational impact our roles call for, and many of our fellow principal engineers expressed feeling overwhelmed. Some were embedded so deeply in their teams' day-to-day execution that they had little time left to pursue the larger, strategic initiatives that

define success at this level. Others had the right scope and support, but found themselves in unfamiliar territory: the skills that had propelled them to this point were no longer the ones that would sustain their effectiveness.

What was missing was not talent but structure: a common understanding, practical techniques, and a concrete path forward. We set out to build that. The result is this book, which was originally designed as a training program for Zillow's principal engineers as they take that next career step and expand their influence by driving business impact.

While the content began as Zillow-specific, we've worked to generalize it for a broader audience—preserving the practices and strategies that resonate across companies and contexts. We've adapted examples, clarified terminology, and removed internal shorthand where needed, but the core material remains grounded in real problems we've faced. Our goal is to share practical insights and hard-won lessons from that journey—shaped to support engineers navigating complex technical and organizational challenges.

By the time of publication, over 90% of our principal engineers—ranging from those new to the role to those with 20+ years of experience—have completed the program. Feedback has been overwhelmingly positive. Participants have praised the program for providing practical techniques, establishing a shared language, and fostering leadership and vision.

> *"I wish something like this had been offered when I was first promoted to principal."*
>
> *"The program was great to frame the context of expectations and empower principals to lean into the role, specifically in the aspect of leadership and vision."*
>
> *"I think this has been the most valuable Zillow course I have taken so far and I feel that I can apply the learning material in this course directly to my day to day activities."*
>
> *"Lots of new good techniques I wasn't familiar with and things I was already doing which I guess come naturally being a principal, but was still nice to get a reinforcement that I was on the right track"*

*“I would encourage every principal engineer to go through this.”*

We’re sharing this content with the wider industry because we recognize how valuable it would have been earlier in our own careers. Whether you are just stepping onto this path or have walked it a long while, we hope it offers you practical insights and valuable guidance on your career journey.

# How to read this book

We've designed this content to flex around your needs — whether you want the full principal engineer learning journey or just a quick dive on a specific skill. Here's how to navigate.

## Our intended audience

This book is primarily written for principal engineers. If you're in this role and seeking ways to amplify your influence, navigate ambiguity, and grow in your career — this book was written for you. It may also be useful to senior engineers considering a transition into a principal role, or to engineering managers who want to better support the principal engineers they work with.

Note that we use the term *principal engineer* in this book, as it's the term we use internally at Zillow®. However, we recognize that *staff engineer* is also widely used across the industry, and we make no distinction between the two. The content should be equally applicable regardless of formal title.[1]

## How it's structured

We've organized the book into five distinct parts. You can jump in anywhere, but the skills build on each other from part to part.

**Part I – Creating impact through leverage**
Learn how to position yourself where your effort multiplies, generating outsized impact.

**Part II – Cultivating relationships**
Master the connective tissue of any engineering org — building trust and collaboration across levels.

**Part III – Leveraging influence**
Shape outcomes and attitudes without formal authority, by accruing trust, demonstrating expertise, and using clear communication.

[1] Zillow® is a registered trademark of MFTB Holdco, Inc.

**Part IV – Scaling yourself**
Maximize your impact as a principal engineer without burning out your limited time, energy, or attention.

**Part V – Building technical strategy**
Craft a technical strategy that balances immediate deliverables with a long-term vision to meet business goals.

In each part, you'll find focused explorations of key themes in sections like:

**Techniques**
Actionable patterns you can apply to your day-to-day work.

**Scenarios**
Fictional Zillow-style examples that demonstrate concepts in the work environment.

**Exercises**
Reflection prompts and hands-on challenges to help the learning stick. Use them to reinforce what you learn or skip them if you prefer!

**Antipatterns**
Common pitfalls to avoid.

You can treat each part as its own mini-course: pull the techniques, scenarios, or exercises that address the challenges you face today. Skip to the part that meets your needs, or follow in sequence to see how the concepts layer and reinforce one another. Whatever path you choose, you'll learn great stuff, so the choice is yours.

## Facets and the engineering ladder

Throughout this book, you'll see excerpts quoted from Zillow's Software Development Engineer Leveling Guide referencing a particular "facet." The leveling guide organizes role expectations into seven of these "facets," such as code, architecture, and leadership. We've provided our complete guide as an appendix. While reading it isn't necessary to engage with the content of the book, you may find it helpful to refer to.

## Zillow’s core values

Along the way, you'll see references to Zillow's core values like #OwnIt, which shape how we work together, serve our customers, and make decisions every day. You can read more about them in the relevant appendix.

## Cloud HQ

While we believe the insights in this book are broadly applicable across many tech companies, there’s an important aspect that makes Zillow different: our fully remote work environment, affectionately-termed *Cloud HQ*. This is a foundational strength for our company and a core part of our identity that affords our employees autonomy and unparalleled focus. It does, however, place greater responsibility on our principal engineering community, requiring enhanced connection, coordination, and relationship-building. This fact is reflected throughout the book.

**Part I**

# Creating impact through leverage

> *"Impact is paramount, and you must scale. Sometimes, your impact will be greatest when you go deep to author critical code or uncover a subtle flaw; at other times, your impact will be greatest when you act broadly to shape approaches across several teams or systems."*
>
> *— Zillow Leveling Guide*

In this part, we'll explore what leverage means for principal engineers. Like a physical lever amplifies force to move a heavy object, leverage in engineering amplifies your impact. It's about using your knowledge, skills, and influence strategically to drive outcomes that extend well beyond your own work. This means making choices that create value across teams, systems, and time horizons. We'll delve into techniques for thinking long-term, creating clarity from ambiguity, and empowering others — skills that help you maximize your leverage as a Principal Engineer.

**Section 1**

## Thinking long-term and designing for change

> *"You seek to build and cultivate* capabilities *over* features*: you understand that a curated set of fundamental strengths enables a great variety of particular features, quickly and*

| |
|---|
| *affordably. You build systems that anticipate change; you enable an architecture where it is easy to disband and replace particular systems when the need arises."* Architecture facet |
| *"You are an expert in your business domain, shaping technical investments to unlock long-term business capabilities."* Strategy facet |
| *"The feedback cycles are longer than before, stretching across quarters and into years — a solution must not only deliver incremental value, but withstand the test of time via resiliency, scale, and adaptability."* Strategy facet |

*— Zillow Leveling Guide*

As a principal engineer, it's your responsibility to align long-term technology direction to long-term business needs. You must reason over longer time scales than more junior engineers — that means thinking across years, not just quarters.

Businesses are generally good at evaluating short-term value: what a solution delivers compared to what it costs to build. It's natural to try to maximize the value of the former and minimize the latter in a kind of basic equation:

$$Value = features\ delivered\ by\ the\ system - build\ cost$$

Even early in our careers, we can recognize that this view is too simplistic. A solution exists for a period of time, and during that time, the utilized features accrue value. But every solution also carries a maintenance cost, which we can generally refer to as debt. Like financial debt, deferred maintenance tends to grow over time, accruing unwanted long-term interest. So we get a more realistic equation, something like:

$$Value = \int feature\ utilization - \int debt - build\ cost$$

However, we also know that most solutions will have to evolve to meet changing business needs. A system that can easily adapt is better than one that is expensive to change. So an equation that accounts for adaptability looks like:

$$Value = \int feature\ utilization - \int debt - \int cost\ to\ change - build\ cost$$

Note that build cost is constant — therefore, over the lifetime of the system, it is (likely) dominated by the cost of change. Adaptability, in this view, is more important than initial build cost.

Yet, we must recognize that build cost is immediately accrued while the value of adaptability is speculative: the future success of associated features/products isn't guaranteed, so any associated ability to grow and adapt may never be realized.

This, of course, is the challenge that a principal engineer faces: to inexpensively build a system that, over time, will prove adaptable to the changing needs of the business. To replace ambiguity with short-term deliverables aligned to long-term vision is the primary role of the principal engineer.

## **Technique:** Working backward

A proven method for marrying vision to short-term value is to work backward from a desired end state.

This end state may be an idealized product experience. Or, it may be an ideal system state — some combination of adaptable, scalable, resilient, and economical. Whatever this desired end state is, write it down and make it explicit. Describe why it matters to the business and what value it will create.

Use this document to build alignment. Coauthoring your end state with stakeholders can be powerful, increasing buy-in and the sense of shared commitment. Not everyone will agree on every detail — or even that the entire vision will be built. But if you can agree that the vision is desirable, it becomes a basic bias around which you and your stakeholders can orient all future decisions: in general, a decision that advances the system toward the stated vision is preferred to one that doesn't.

With this vision established, work backward from there.

- What major system components need to be built?
- How robust or fully featured does each need to be?
- Which pieces should be built as distinct services or components early, because they'll be hard to split out later?
- Which pieces will be easy to refactor at a later point?

Write this down as you go, establishing a rough sequence of work. Identify what constitutes a first iteration — where is the cut line for the initial deliverable?

Rarely will the first iteration include only simple, easy-to-deliver work. Often, you'll want to de-risk a larger project by working as early as possible on the most ambiguous 'big rocks.' Such as, investigating whether a core architectural assumption will hold true. Or, identifying that a particular framework may have a high initial adoption cost, but will be far more expensive (read: impossible) to later retrofit into a project.

This is key. As the principal engineer, it's your responsibility to identify the major design risks and advocate for working on them early. You don't build a house by assembling a bedroom first, no matter how sleepy you are. The work you put into clarifying priorities, building consensus, and anchoring the team in shared principles creates the trust and mandate you need to address critical risks early — that's your leverage.

To illustrate, let's consider a scenario. We'll return to this scenario throughout this part to explore different techniques and approaches. Though rooted in the Zillow product, it's a fictitious scenario and, like all scenarios in this book, isn't intended to map to real world systems, people, or organizations.

**Scenario**

Suppose that you are a principal engineer working on the home shopping experience. The product organization is proposing to add a new feature that allows a customer to share a favorited home with a co-shopper, such as a partner or friend.

While the initial request is simple — share all of a customer's favorite homes with one other person — you know that the B2B side of Zillow is interested in enabling a customer to better share information like favorited homes with industry partners, such as agents and loan officers. You recognize that you and your fellow engineers in the B2B side of Zillow have a common authorization problem — and you know that modeling authorization is a very difficult problem, fraught with edge cases and complications.

Currently, the products for customers and industry partners are disjointed: the easiest thing to do is to build in isolation. After all, no one in the business is advocating for integration — yet.

To get ahead of the problem, you work with fellow principal engineers from across the company to articulate a vision: a common standard for describing authorization rules. Perhaps this happens asynchronously; perhaps the issue is contentious enough that it warrants a working onsite.

Regardless, you coauthor a paper that describes an end state where future integrations will become doable because the standard is shared — an authorization rule written for one stack will be fundamentally interpretable by another.

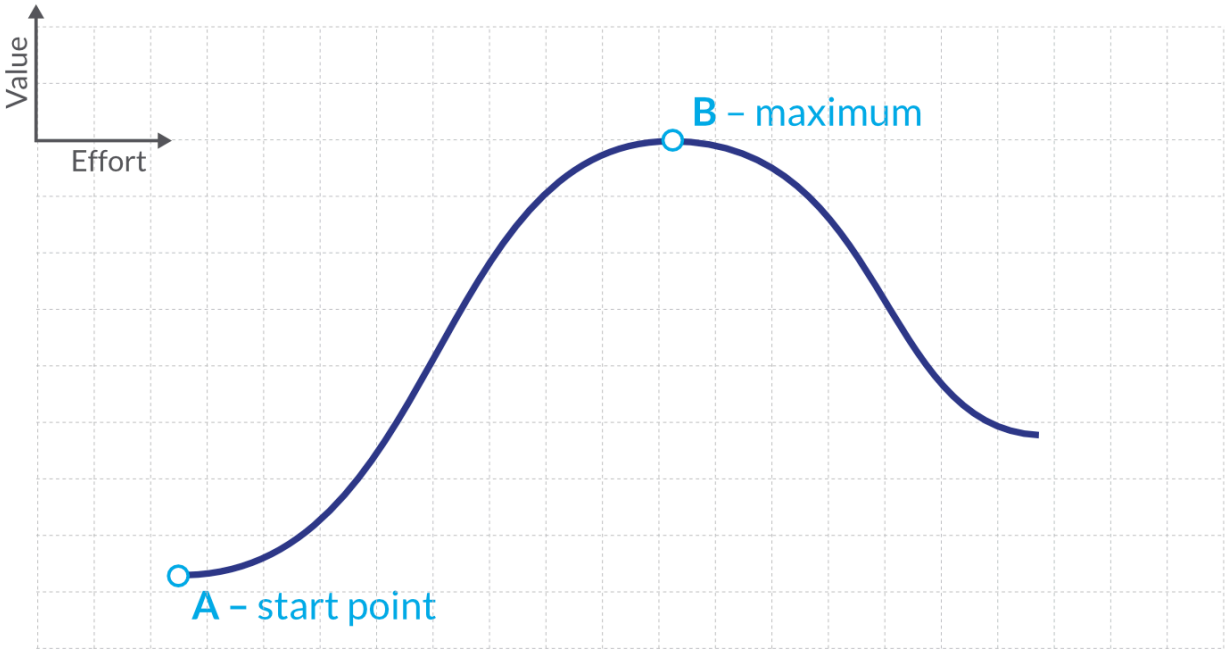
There is talk about the value of building out a centralized solution to store the authorization rules. This will require significant investment and take time. One of your B2B partners has a partially built but non-generalizable solution, and they cannot wait. Others, however, can wait or use easily-reversible stop-gap options.

So, you agree that your team will evaluate backend stores with an eye towards building a generalized solution. While the B2B team will continue forward with their existing solution to meet deadlines, they agree to adhere to the authorization standard. Future integration — while far from trivial — will be viable because of these agreements.

This vision established, you can now turn to your particular problem — authorizing one shopper to share data with another. You know that choosing your datastore will be important, though, so you need to allocate time to test and design (we'll pick this up later in the "Creating clarity from ambiguity" section).

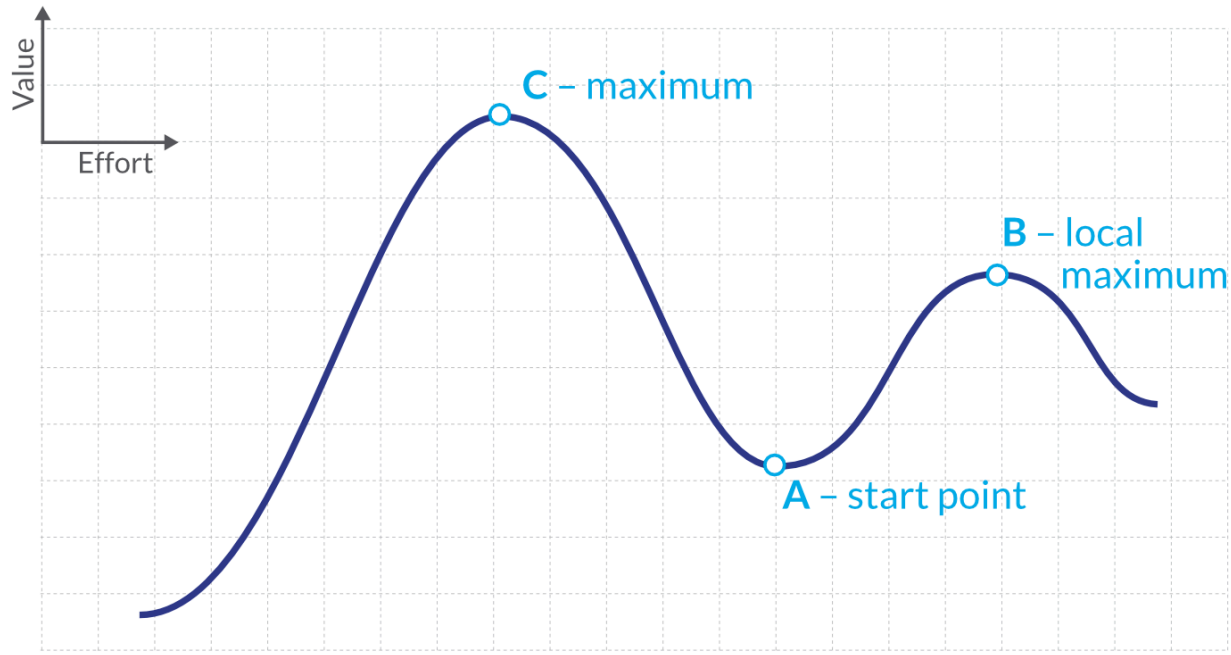
## **Technique:** Beware local maxima

Another way to think about working backwards from an end state is to frame it in terms of avoiding the well known trap of optimizing for a local maximum.



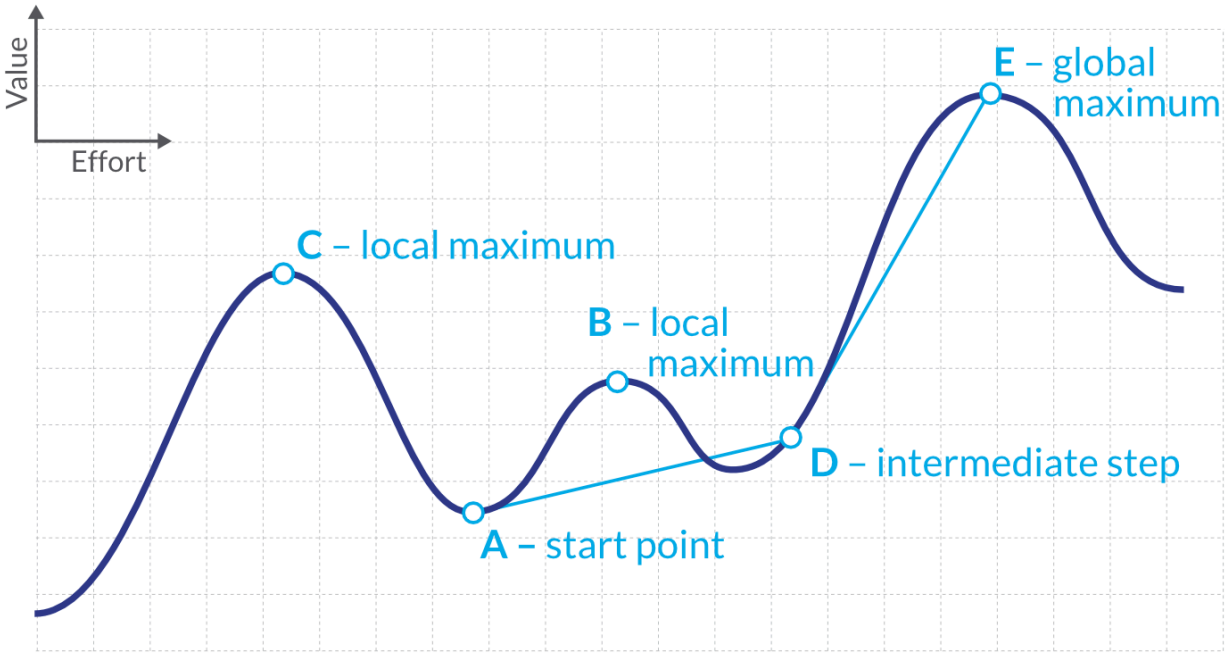

Starting from point A, we can iteratively develop our approach via small changes to arrive at the optimal configuration, represented on the diagram by point B. This is a common conceit in iterative system development: *We'll keep making small changes until the system is optimized.*

Let's zoom out.




If we broaden our view to consider more possibilities, we might see that iteration in a *different* direction (towards a different architecture, technology, etc.) could result in a better outcome (point C). From this perspective, we see that point B was a local maximum, not a global one.

But let's zoom out even further.




Point E is the best configuration — but it lies beyond B. We could iterate to B, but moving forward will be high effort— and we'll have to suffer *lost* value if we keep slowly iterating along the curve. In practice, this may mean making significant, nonfunctional changes to affect a major change — pragmatic, but slow and expensive.

However, if we start by working backward from E, we might chart an alternative path: from Point A to D to E. We don't chase B — we skip right past it. The initial value added from A to D is lower, but it may quickly position us to move up the curve to E.

By thinking long-term and working backwards from an end point, we can avoid the trap of chasing a local maxima that inhibits, rather than enables, the business. In this way, we create better outcomes without requiring additional engineering resources — maximizing our individual impact by charting a better path.

| Exercise | Let's pause and reflect on thinking long-term. In your career, when have you had success driving long-term thinking? What techniques did you use?<br><br>On the flip side, were there times where an attempt to think long-term failed in the face of other pressures? Why? Do you think any of the techniques described could have helped? If not, what else might have helped? |
|---|---|

## Section 2

# Creating clarity from ambiguity

> *"You are expected to identify, define, socialize, and break down novel problems in your organizational domain."* Strategy facet
>
> *"Your design documents are models of reasoned decision making, persuading the reader to accept a conclusion through the careful specification of the problem, thorough presentation of context, and rigorous analysis of trade-offs. You use data to bring clarity to contentious issues."* Communication facet

*— Zillow Leveling Guide*

As a principal engineer, it's your job to peer further into the technical murk than others. When the business launches a new initiative and no one knows where to start, you begin shaping that ambiguity into a concrete plan. You turn high-level ideas into architecture — sketching out systems, responsibilities, and flows. And when the initiative moves forward, you assess whether the many parts are solving the problem in the correct way, or if we're missing an opportunity to approach the problem differently. Turning the broad wishes of the business into actual software is part and parcel of what we do.

Yet, it's also more than that. A principal must proactively survey the ambiguity around them — things like changing technologies, aging architectures, unanticipated business needs, the shifting behaviors of dependencies and clients — and spot the emergent problems before anyone else.

Then comes the hard part: helping others understand the problem. Not only do you need to see the issue, you must explain it clearly so others grasp the implications. It isn't enough to envision the problem in your mind; you must persuade others.

In other words, some of your highest-leverage impact will come less from solving problems than from defining them. By helping others understand the size, scope, risk, and trade-offs of various approaches, you reduce ambiguity and accelerate better decisions.

**Scenario**

Let's return to our scenario where you worked with other principal engineers across the company to articulate the problem: a generalized approach to authorizing data access would, in time, enable a more integrated product experience. Your ability to articulate the problem to get others on board was key to making progress.

Clarifying the problem is a process itself and can unfold over days, weeks or even months depending on the circumstances, through an approach like this:

1. Speaking with individuals, probably other principal engineers, line managers, and product managers across the company. You leverage your network to identify folks to socialize the problem quickly with like-minded thinkers.
2. After initial feedback suggests that the rough contours of the problem are well understood, you articulate the problem quickly in a one-page document. You use this to socialize the problem further, especially with leadership.
3. You work with your leadership to schedule a working onsite. Several difficult problems need to be whiteboarded, and this will happen much more efficiently in person. You can now pick a date to work backwards from: after all, this cross-cutting authorization problem would be easy for any one individual to deprioritize. By forcing a get-together date, you drive the group towards a decision.
4. You translate the decisions from the onsite into a compelling six-page document, cosigned by your fellow principal engineers from across the organization.
5. You then take this document to other forums for socialization and feedback.
6. Note that this work didn't necessarily block juniors from other kinds of work in the same problem space. As a principal engineer, you context switch frequently — you can drive this thread of work while helping move other aspects of the project forward.

The act of identifying a problem does not, in and of itself, create impact. It's not enough to notice the iceberg, you need the captain to actually turn the ship away

**Antipattern**

A PoC must teach us something about our product or UX thinking, or about our engineering approach and the feasibility of a particular implementation strategy. There's little value in converting a well-crafted UX mock into a working demo unless it's geared toward specific educational objectives.

from the collision! By clarifying the size, scope, risks, and trade-offs through conversations with fellow principal engineers, line managers, and product partners—and socializing those insights through documents and working sessions—you help map out how to steer away from the collision.

## **Technique:** Proof of Concept

When you're evaluating a new pattern or technology, hands-on work is often the fastest way to demonstrate viability. A working prototype — or proof of concept — can do more than a dozen pages of design. It's one of the best ways to author high-leverage code and break through analysis paralysis by tangibly demonstrating the feasibility (or infeasibility) or a particular idea. It can facilitate constructive discussion of an otherwise abstract or novel approach to a problem. A proof of concept (PoC) is not an MVP (minimum viable product). It's an artifact meant to test a hypothesis — 'nothing more. It's intended to be fast, cheap, and disposable. And because it won't be released to production, you're free to discard the typical rigor you'd apply to your code.

Remember: a PoC doesn't replace an eventual design. It lets you collect information through hands-on learning — and those learnings should be captured in a proper design document, where they will inform a wider range of considerations.

PoCs are tools for learning, not shipping.

### Scenario

With a company-wide authorization approach now in place, you turn your attention to your own domain: co-shopping. There is a new backing datastore technology that specializes in modeling granular authorization rules. The website is slick, you've heard good things from others in the industry, and the documentation looks promising.

**Antipattern**

Don't confuse two-way doors with biasing toward the cheapest or quickest solution. Too often, a solution chosen in haste to meet an arbitrary deadline is the most difficult to unwind — e.g., an ill-thought-out API adopted by dozens of dependencies or a datastore choice that's difficult to migrate away from. Be intentional and design for reversibility, especially when ambiguity is high.

You could draw up a spreadsheet to begin a feature-by-feature comparison with alternative technologies. This might be useful, even necessary. However, you also know that this is new territory for yourself and the larger organization — no one has used a technology quite like this one before. You want to get a sense of how it works and how easy it is to use. So, you roll up your sleeves to build a proof of concept.

You know you're not building an MVP. You're testing for specific factors, like: How well will this integrate with GraphQL? How difficult is it to unit test against? Is language support consistent for our Golang, Kotlin, and JS backends? You build out one or two demo-able examples. These will quickly build confidence in your approach — or signal that you need to try something else.

A quick PoC gives you clarity — and gives the team confidence to move forward or change direction, together.

## **Technique:** Seek two-way doors

A two-way door is a decision you can reverse easily or inexpensively. Walk through it, and if you decide you don't like what's on the other side, step back through and make a quick exit. A one-way door is the opposite:  once you go through, you'll struggle to claw your way back out. You can minimize risk and make decisions more quickly by identifying and favoring two-way doors.

Two-way doors help you avoid analysis paralysis. If you're stuck between several design options, pick the one that is not too expensive to undo later. Start building toward it, and learn from the work along the way. Junior members of your team can make progress in a direction while you (or other leaders) continue to analyze and learn. This might increase throw-away work — and that's fine! Code isn't intended to live forever. In general, when you have multiple options and the trade-offs among them are hard to quantify, bias toward the one that is easiest to back out of.

**Antipattern**

Watch out for poor or irrelevant metrics. Focusing on small or short-term fluctuations can leave you chasing local maxima. Sometimes the most important metrics are hardest to measure; don't simply choose the metric that's most readily available. Choose the one that tests your convictions.

This is a technique for minimizing risk in achieving the vision, not minimizing the effort or time required. If upfront analysis or authoring a proof of concept isn't possible, reduce risk by choosing the approach that can be reversed. Take a step, gather information, and adjust. That's how to reduce risk and create clarity from ambiguity.

This approach calls for something more nuanced: work that maximizes learning new, valuable information — even if you don't keep the code — may be the best way to move your long-term vision forward.

## **Technique:** Quantify the problem

A great way to reduce ambiguity is by distilling the problem to a set of quantifiable metrics. Ideally, these are well-aligned to business value, so stakeholders of all stripes can easily understand what changes to the metrics mean — e.g., more home shoppers getting connected with an agent is a good thing, fewer people scheduling a tour is a bad thing, etc.

Sometimes we'll struggle to tie engineering work directly to a clear business metric. We might be tempted, in those moments, to throw up our hands and say, "trust me." But that won't help others understand the problem, and, even if your effort proceeds, you may find yourself struggling to articulate the value you delivered at the end.
Instead, try to quantify the problem, however imperfectly. Be upfront about any gaps, and trust your audience to listen. Too often, we worry that others will chase a single metric aggressively and at the expense of other measurements, so we avoid quantifying anything at all.

Look for existing frameworks and industry standards. Borrowing quantification measures from proven sources adds credibility and saves time. For example, Zillow uses Core Web Vitals to measure the performance of our webpages. It is easier to convince others to adopt  widely-accepted standards and they're often more reliable than custom metrics.

Metrics reduce ambiguity and help others see the value you see.

**Scenario**
Let's suppose there are two backend storage technologies that you're considering for the co-shopping authorization store. You want to evaluate these two options on three dimensions: scalability, performance, and resiliency.

You build a shared testing framework and benchmarking process. It simulates load and gathers metrics.

- Scalability: CPU utilization under different load patterns
- Performance: Average latency
- Resiliency:  Latency sensitivity when adding or reducing nodes in a cluster while under load.

All together, you gather a set of metrics by which to compare and contrast solutions. Will this capture everything? No, but it will help you make a more informed decision.

## **Technique:** Act decisively

You will face situations where the right decision is not obvious: no clear metric exists, no clear consensus holds. In these moments, your job is to to #OwnIt — use your best judgment and choose a path forward.

Analysis paralysis is a real problem — we seek certainty where no certainty can be found, burning precious time and increasing our chances of failure. One of the best ways to create clarity is to give the team a path forward. Your decision will lead to action — detailed designs will be written, PoCs will be created, code will get pushed — and action will lead to learning. You might learn you made the wrong decision, and you need to back-track. This is a good thing — hopefully, you anticipated this and chose a two-way Door. The information you've gathered will help you choose a next-best approach — and often at much lower cost than waiting for perfect clarity would've required.

**Scenario**
Let's return to the situation we left. You're considering two alternative technologies. The first has the quantitative edge, performing better across a greater set of metrics—though not across all. The second is easier to use. Furthermore, the second is a well-maintained open source project under active development with multiple large, institutional users and an associated SaaS

vendor. The first, while still updated and supported, doesn't appear to have the same industry momentum.

There is no clear choice. Others are suggesting different dimensions to investigate, write up, and attempt to quantify (or, at minimum, assess). You recognize that delay puts the larger project at risk. Product might reasonably call your effort to tackle the generalized authorization problem scope creep and push for a quick fix — like just  hardcoding a special co-shopper identity into all of the systems.

So, you use your judgment. You decide that the active development and better institutional support of technology two suggests that it will close the performance gap over time — and if it doesn't, performance is good enough for the next 18 months. You write up your reasoning, quickly share it with stakeholders, and close out the technology comparison. Time to move on.

Use your judgment, act decisively, and document why.

| Exercise | Let's pause and reflect on creating clarity from ambiguity. In your career, what successes have you had helping others — other engineers, leadership, cross-functional roles — better grasp a problem and build towards a better solution? What challenges did you face?<br><br>Have you leveraged any of the techniques presented here? If so, what worked and what didn't? What other techniques have you found helpful? |
|---|---|

**Section 3**

# Creating surface area for others

> *"You seek excellence in all things, such that your systems are archetypes of flexibility, maintainability, security, and testability. You understand that your impact scales with your influence: you cannot uphold these standards solo, but instead must rely on the team to embody them even when you're not looking."* Leadership facet
>
> *"When you identify workstreams in large projects, you guide juniors to effective solutions and inspect their estimates."* Leadership facet

*— Zillow Leveling Guide*

You must work through others to translate insight and intent into action and outcome. This is how you scale your impact. Yet, it's not enough to just hand requirements to others, dust off your hands, and walk away — you're responsible for overseeing the implementation of your ideas. You must learn to inspect the work of others, often at a lower fidelity than you'd like, and provide high-quality feedback.

A principal engineer enables others to do good work by creating the surface area they need to move. Start by breaking down a large project into several subparts, each actionable by a smaller team or individual. It may seem straightforward, but the way you delegate tasks can make or break your impact at scale. If you tell someone how to connect every dot, you'll get the outcome you planned — but you'll stay in the loop forever. If you guide someone through solving the problem themselves, you free yourself up to focus on what's next. To put a twist on the parable: give an engineer a design, and they'll write some code; teach an engineer to design and you can go fishing.

In this section, we'll explore techniques to delegate effectively and guide work at scale.

## **Technique:** Know the engineers of your organization

This may sound simple, but it's essential to the principal role: you are a leader of fellow engineers, and you must understand their strengths, their weaknesses, and

their ambitions. While you are not a people manager, you must learn to work closely with and think like a people manager: Who is trusted to deliver what? Who desires to grow, and along what dimensions—and so might be uniquely motivated to tackle a particular problem? Are there weaknesses that require additional guardrails or support to be successful?

No matter how brilliant a design or a project plan, if it can't be executed by the team it's useless. Similarly, by tailoring work to a team's capabilities, you might find that your organization can deliver much more than you realized.

**Scenario**

Let's return to our scenario and rewind a bit, back to when you were first considering two PoCs to evaluate datastore options. Instead of writing these PoCs yourself, you recognized you could let others do so. This approach helped the project scale in two ways: it freed you to steward other parts of the project, and it reduced risk by giving the work to junior engineers who had more time to dig in and think carefully.

You still wrote the benchmarking harness. This ensured that both PoCs were evaluated consistently. You reviewed code and asked the junior engineers to make qualitative evaluations as well.

Before the junior engineers' work began, however, you spoke to the relevant software development manager (SDM) about who would pick up this work. The SDM suggested two senior engineers, Veena and Ryan. You've worked with both extensively, and noted their strengths. Veena is deeply curious and quick to try new technology, but with a usefully skeptical attitude. She's a great fit for PoC work. Ryan, on the other hand, tends to take a cautious and systematic approach — a great asset when a complex change to a distributed system needs to be thought through, but a potential risk for quick PoC work.

Let us suppose you raise your concern with the SDM, who suggests you mentor Ryan during the process. You note that while that might work, this may not be the best circumstance for mentorship: the deadline is tight and you're already risking over-committing your time. Mentoring in this kind of circumstance might take longer than just doing the work yourself. Instead, you accept a minor delay to the calendar and ask that Veena do both: this gives her the opportunity to do a more complete side-by-side qualitative evaluation herself, which will help her grow.

## **Technique:** Inspecting, intervening, and letting others fail

Junior engineers are humans, and humans learn best from their own mistakes. If you want to cultivate an organization that adapts and grows at every level — and produces ever-better outputs at every scale — you must give individuals the autonomy to think and act independently.

This comes with risk. The cost of learning through mistakes may have intolerable consequences for timelines or requirements. If we identify a mistake early and intervene, we might save the company time and money; yet, if a principal identifies all the mistakes themselves, they will build a brittle organization.

To navigate this tension, we must be intentional about how we create safe spaces for failure, how we inspect, and when we intervene.

**Creating safe spaces for failure**. A mistake can take many forms, such as a messy but functional change that complicates the code base, a bug that requires a rollback in production, or a brittle architecture that frustrates feature development for months or years to come.

When you encounter one of these situations, think once again in terms of two-ways doors: Which failures are quickly reversible and which are not? Exercise your judgment: Which decisions or actions warrant your attention because they will be difficult to unwind? Intervene in those, ignore others that are trivial to unwind, and keep an eye on the more ambiguous.

You can also cultivate safe spaces for failure by investing in automated guardrails or self-healing mechanisms. A bug in the code that's discovered by a failed unit test will help a junior learn without being of any consequence to the larger organization. A bug discovered in production and rolled back automatically may impact timelines, but will have limited impact to customers — and coupled with strong organizational processes (operational reviews, root cause analysis, etc.) this experience will provide a great opportunity for juniors to learn.

**Inspection**. To determine which decisions require your intervention, you need a way to discover which decisions are being made in the first place. You must

inspect. However, you must be careful to ensure that the process of inspection doesn't unnecessarily remove autonomy and undermine learning.

Avoid becoming the “chief inspector” of your organization, where each engineer prepares their work to be inspected by you. They will come to know your preferences and idiosyncrasies, over-indexing on your personal approval and opinions. This is an easy trap to fall into.

Rather, encourage a culture of inspection by all. An idea disagreed with and discussed early is a potential lesson learned — and a shared decision by juniors is an opportunity for all to learn. Model this behavior in your work as well, seeking feedback not only from your own peers but from juniors — and be open about your mistakes. Seek to make inspection a shared cultural norm, and subsequent failures are more likely to be communally shared and lessons communally learned.

Finally, lean into standard rituals that create safe spaces for feedback. Conversations at the whiteboard can offer opportunities to *suggest* alternative approaches or *hint* at potential failure modes — reducing the risk of failure without prescribing a path. Design documents can have a similar function. Don't save these for major design work alone — even the simplest change offers an opportunity for a junior to take ownership of an analysis, propose options, and receive feedback early. If a shared culture of inspection is thriving, you may not have to review many or most of these documents yourself!

**Intervene**. There will be times where it is necessary to course correct. The challenge here is to time these interventions appropriately. An expensive diversion may be best course corrected through a conversation early in a design process — here, open conversations around the whiteboard or in 1:1s offer a low-friction method of intervention. A culture of design documents gives you visibility into decisions happening across the organization. You may identify a risk that others have missed and intervene on a particular decision or design, while quietly letting other lower-risk initiatives proceed.

The same holds true for MRs. While you will need to be intentional about the MRs you review in order to scale yourself, reviewing MRs is an important tool in your toolbox. You’ll personally review critical changes or changes to particularly complex paths. In less critical cases, you may informally review MRs to

double-check that a culture of inspection is flourishing: Are the right questions being asked and addressed?

### **Technique:** Lead with questions, not answers

Well-formed questions can prompt introspection and creativity. They help others think more deeply and explore more broadly.

Questions help you pause before jumping in with “that won’t work” or “try this instead.” Telling someone what to do might get the right outcome, but it doesn’t grow the right skills. Asking thoughtful, patient questions guides your team and helps them arrive at answers on their own. That’s what builds stronger engineers — and a stronger team. When a person arrives at a conclusion themselves, it’s theirs — they’ll have conviction and they’ll be more likely to drive it forward independently.

Questions also give you a chance to learn. You won’t have time to master the details of every technical domain you’re responsible for — and that’s okay. Rely on your term and learn from them. That’s how you scale your impact. So, avoid jumping to the conclusion or asking leading or loaded questions: ask questions that help you genuinely understand how juniors are conceiving of and approaching the problem.

Along with being a more inclusive leadership approach, the act of asking questions helps reduce the chance of incorrect assumptions — both those made by junior engineers as well as any assumptions you might be making. Often, the best questions to ask are the most fundamental, as they force everyone to re-assess assumptions. You, as the senior, will be best positioned to question the fundamentals.

Tailor your questions to the audience. A senior team you know well may prefer direct questions, and not feel intimidated by your title. A more junior group might respond well when you offer a wrong answer — giving them something concrete to correct and disprove.

| Exercise | Reflect on these techniques from the perspective of your current responsibility. How are you creating surface areas for others? What techniques are working? What challenges have you faced? |
|---|---|

Part II

# Cultivating relationships

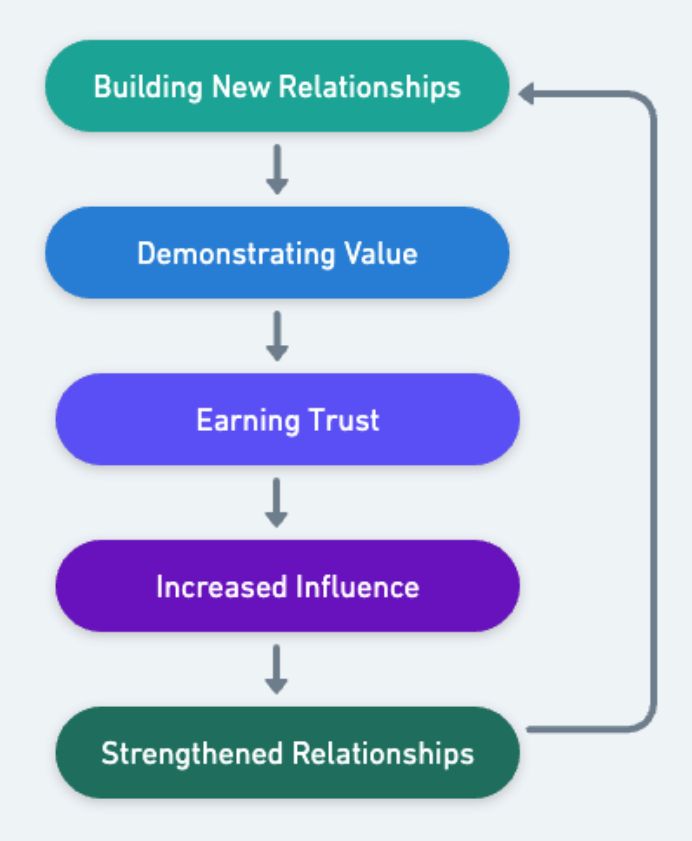
Technical expertise is expected of a principal engineer. What sets you apart as a truly exceptional principal is your ability to cultivate meaningful relationships across all levels of the organization. Relationships are the connective tissue of any engineering organization, enabling seamless collaboration, fostering innovation, and driving successful project outcomes.

In the complex world of software engineering, where socio-technical systems are deeply intertwined, building and maintaining strong relationships is not just beneficial — it's essential. The interplay between social structures (like teams and communication networks) and technical systems (like software architecture and tools) means that success depends on more than just individual skill. It hinges on how well these systems work together, which is fundamentally a relational challenge.

## The correlation between relationships and influence

In parallel with building relationships, your ability to influence others is a key determinant of your impact. Influence, which will be discussed more deeply in part three, isn't about formal authority; it's about earning trust, demonstrating credibility, and building a network of allies. Strong relationships are the foundation of this influence. Whether you're advocating for a particular technical approach, driving a cultural change, or rallying support for a new initiative, the relationships you've cultivated across the organization give weight to your voice.

Forging relationships gives you more opportunities to demonstrate value and earn trust, which increases your ability to influence. This in turn strengthens existing relationships and opens doors to new ones, creating additional trust and influence.




By mastering the art of building and nurturing relationships, you can expand your influence, drive greater alignment across teams, and contribute to a more cohesive, innovative, and resilient organization. This part will guide you through practical strategies for cultivating these essential relationships, helping you to not only advance your own career but also elevate the entire organization.

## Building and understanding relationships

> *"People don't care how much you know until they know how much you care."*
> *— John C. Maxwell*[2]

As a principal engineer advances in their role, the ability to build and maintain strong relationships becomes increasingly vital. But understanding others and forming meaningful connections can be challenging, especially in complex and fast-paced work environments. We'll explore two areas that are essential for relationship-building: understanding the common barriers that prevent us from

[2] John C. Maxwell, *Relationships 101* (Harper Collins Leadership, 2004)

truly connecting with others, and recognizing the signs that indicate a strong, healthy relationship.

## Common barriers to understanding others

Understanding others is a crucial part of building meaningful relationships, yet it’s something many struggle with. Here are some reasons why we might fail to truly understand others.

1. **Self-centeredness:** One of the biggest barriers to understanding others is being too focused on ourselves. When we're more concerned with our own needs, opinions, and perspectives, we miss the opportunity to genuinely see and understand those around us. Building effective relationships requires shifting the focus from "me" to "we."

2. **Making assumptions:** We often make assumptions about others based on our own experiences, backgrounds, or biases. These assumptions can lead to misunderstandings and prevent us from appreciating the unique perspectives and experiences of those we interact with. Overcoming this requires intentional listening and an open mind.

3. **Lack of active listening:** Many people listen with the intent to reply rather than to understand, which leads to shallow interactions and missed opportunities to connect. Active listening — where you fully concentrate, understand, respond, and then remember what is being said — is crucial, especially in collaborative environments like video meetings where distractions from Slack messages or emails are common. If you can understand the motivations behind what someone is saying and identify what the other person is trying to achieve, you’re in a better position to align goals and build stronger relationships.

4. **The need to be right:** The need to always be right or to dominate conversations is another major obstacle to understanding others. Relationships thrive when we prioritize understanding over winning arguments. Humility and the willingness to learn from others’ perspectives are key to overcoming this barrier.

5. **Not valuing differences:** Differences in personality, background, and perspective are strengths that can enrich relationships. However, if we don’t value these differences, we may struggle to understand others. Embracing

diverse thoughts and seeking to learn from those who think and act differently can lead to deeper understanding and stronger connections.

## Indicators of strong relationships

A principal engineer needs to be able to recognize the signs of a strong, successful relationship. A strong relationship is characterized by trust, open communication, and mutual respect. When you've built a strong relationship, you'll notice several key indicators:

- **Open communication:** The person feels comfortable reaching out to you with questions, concerns, or ideas, and you find that your conversations are candid and productive. Two-way communication is a hallmark of a healthy relationship.
- **Mutual support and collaboration:** There's a natural flow of collaboration, where both parties are willing to support each other's goals and challenges. You'll notice a willingness to share resources, offer help, and work together towards common objectives.
- **Constructive feedback:** You can give and receive feedback openly without hesitation, knowing that it will be taken in the spirit of growth. This exchange of feedback is a sign that the relationship is built on trust and mutual respect.
- **Shared successes:** You celebrate successes together, recognizing that your collaborative efforts contribute to positive outcomes. Whether it's a project milestone, a solved problem, or a team achievement, these shared victories are a testament to the strength of your relationship.
- **Established trust:** Trust is the foundation of any strong relationship. When trust is present, both parties feel confident in each other's intentions and capabilities. You'll notice that there's a strong sense of reliability and integrity, where you can depend on each other to follow through on commitments and be honest in all interactions. This established trust allows the relationship to flourish even in the face of challenges.

Building and maintaining strong relationships is essential for your success as a principal engineer. Overcoming common barriers like self-centeredness, assumptions, and the need to be right is foundational to creating trust, open communication, and mutual respect. Recognizing the signs of strong relationships

— such as collaboration, constructive feedback, and shared successes — helps you gauge your effectiveness in building connections. As you move forward, remember that understanding and connecting with others is an ongoing journey. The next sections will equip you with practical techniques to enhance these skills and create lasting, impactful relationships.

Section 1

# Building relationships with juniors

> *"You demonstrate effective verbal and written skills, communicating with everyone from the junior-most engineer to the senior leaders of your organization."* Leadership facet
>
> *"You cultivate collaboration, listening to and integrating feedback across all levels."* Leadership facet
>
> *"To your team(s), you set a tenor of excellence: you are skilled at upholding a high-quality bar without alienating others, seeking to cultivate and mentor."* Leadership facet

*— Zillow Leveling Guide*

Your mentorship can significantly shape the careers of junior engineers. Cultivating these relationships helps in transferring knowledge, instilling best practices, and fostering a culture of continuous learning. Strong relationships with juniors also creates a more resilient team, where trust and open communication lead to better problem-solving and innovation. Additionally, creating opportunities for juniors to grow and demonstrate their value is essential. By providing them with challenges that allow them to step up and contribute meaningfully, you not only build their confidence but also reinforce the trust within your team. This approach ties back to the principles of "Creating surface area for others" discussed in part one, where giving others the space to showcase their skills plays a vital role in relationship building.

## **Technique:** Building trust

Trust between you and junior engineers is the foundation of a productive team. Trust empowers juniors to take risks, share ideas, and ask questions without fear of judgment, which accelerates their growth and the organization's overall success. Establishing trust is not just about mentorship; it's about creating an environment where open communication and collaboration thrive.

It goes without saying that being known as someone who gets things done and has a proven track record helps establish others' trust in you. While being known as an engineer who can deliver is significant, that alone may not be enough to

**Antipattern**

Inspection is important and should be welcomed by juniors. Juniors who hide problems risk creating problems in your organization, especially in a remote environment. Reward juniors who proactively communicate — and demonstrate effectiveness — with your trust.

build a trusting relationship with the engineers you work with. You'll need to take positive action, too.

**Encourage autonomy:** Encourage junior engineers to take ownership of small projects or tasks. Provide guidance, but give them the space to make decisions and learn from their experiences. In addition to growing the productive capacity of your organization, this is an effective way to show juniors that you trust them to build and operate responsibly. This can significantly boost their confidence in themselves.

**Recognition:** Acknowledge the contributions and achievements of junior engineers, making sure you give credit when credit is due. Public recognition can reinforce their sense of belonging and motivate them to continue striving for excellence.

**Be accessible and approachable:** Make it clear that junior engineers can come to you with questions or concerns without fear of judgment. While some juniors may reach out to you proactively, others may be intimidated by your title. Check in with them regularly to offer support and ensure they know they have a reliable resource and a friendly ear in you. Follow up on requests you've made or topics you've discussed to show that you are interested in their success.

**Show vulnerability and admit mistakes:** Admitting when you don't know something or when you've made a mistake shows that it's okay to be human, which in turn encourages junior engineers to be honest about their own learning processes. One method to create this open atmosphere is to pause the conversation on a topic in which you have little context in order to explain your current understanding back to the junior — since they ought to know more than you on the topic, it gives them the opportunity to correct minor mistakes and misunderstandings you've developed.

**Lead by example:** You must demonstrate the values and behaviors you want to see in your team. Whether it's how you handle mistakes, how you deliver results or

demonstrate competence, or the way you manage pressing deadlines, your actions will set the tone for what is expected and acceptable within the team.

## **Technique:** Informal Mentorship

Mentorship isn't just about structured engagements — it's about recognizing the day-to-day opportunities to mentor and foster growth in your team. As a principal engineer, you're already involved in code reviews, project kickoffs, and technical discussions, but don't forget to also view these moments as mentoring opportunities. Every interaction is a chance to strengthen relationships, and guide and influence juniors — not just in terms of immediate technical outcomes, but in shaping their overall approach to engineering and decision-making.

Rather than seeing these activities as routine tasks, approach them with the mindset of a mentor:

- **Code reviews**: Instead of simply pointing out mistakes, use code reviews to mentor by explaining why certain practices matter, encouraging long-term thinking, and showing how small changes can have broader architectural impacts. This elevates the conversation from code quality to skills growth.
- **Pair programming**: Beyond solving the immediate problem, use pair programming to expose junior engineers to your problem-solving approach and decision-making process. These sessions allow you to mentor in real time, turning collaborative work into valuable learning moments.
- **Project kickoffs**: Kickoffs can be more than just outlining tasks. Mentoring here means giving context, explaining trade-offs, and encouraging others to see the bigger picture. It's an opportunity to help junior engineers understand the "why" behind the work and how strategic decisions are made.
- **Design reviews**: Instead of focusing solely on the technical merits of a design, turn reviews into mentorship opportunities by inviting junior engineers to explain their thought processes. Engage them in discussing trade-offs and implications, teaching them how to approach decision-making with a broader perspective.

Ultimately, informal mentorship is about creating an environment where learning, growth, and building strong relationships happen naturally. By approaching your

interactions and technical discussion as an opportunity to guide, inspire, and connect, you help junior engineers improve not only their technical skills but also build confidence in their thinking and decision-making. By encouraging open dialogue and inviting questions, you'll strengthen your relationships with your team, creating a culture where mentorship and building trust are a seamless part of your daily work.

## **Technique:** Formal mentorship

Formal mentorship is a structured approach to guiding and developing others within the organization. Unlike informal mentoring, which occurs organically, formal mentorship has defined objectives, timelines, and expectations.

When you decide to take on a formal mentor role, start by identifying where your expertise aligns with the specific development needs of the mentee. It's important to engage in mentoring relationships where you can truly add value — if the skills a mentee wants to develop fall outside your strengths, it's better to guide them to a more suitable mentor.

If you're interested in mentoring, proactively communicate with managers in your organization, letting them know of your desire to support the growth of junior engineers. This not only positions you as a key resource, but also strengthens your influence within the organization.

If you work in a distributed environment like we do, be sure to take advantage of face-to-face time with your mentee when you have the chance. If you and your mentee are both attending an in-office event, be sure to schedule some 1:1 time with them. In-person meetings can offer new insights and help strengthen the relationship as it progresses.

**Keys to an effective formal mentorship**

- **Set clear goals**. Begin by defining the purpose of the mentorship relationship. What specific skills or areas of knowledge does the mentee want to develop? What are their short-term and long-term career goals? Clear goals provide direction and help measure the success of the mentorship.

**Antipattern**

It's not just what you say but how you say it. Avoid absolutes like "you always" or "you never," as these can come across as overly critical and raise defenses. Using "I" instead of "you" can make your feedback more constructive and easier to accept. For example, instead of saying, "You never write efficient code," you could say, "I've noticed that there's an opportunity to optimize the performance of this function."

- **Establish a structured plan**. Work with your mentee to create a structured plan that outlines the frequency of meetings, key milestones, and topics to cover. This plan should be flexible enough to adapt to the mentee's evolving needs but structured enough to provide a clear path forward.

- **Create a safe space for open communication**. Formal mentorship should provide a safe and supportive environment where the mentee feels comfortable sharing their challenges, aspirations, and feedback. Encourage open communication and be an active listener, offering advice and support without judgment.

- **Provide regular feedback**. Regular feedback is essential for growth in a formal mentorship. Offer constructive, actionable feedback that helps the mentee improve and build confidence. At the same time, encourage the mentee to reflect on their own progress and identify areas where they want to improve.

- **Review and adjust goals as needed**. Periodically review the mentee's progress toward their goals. If necessary, adjust the goals or approach to better align with their evolving needs and aspirations. This flexibility ensures that the mentorship remains relevant and effective over time.

## **Technique:** Giving feedback

Feedback is a crucial tool for growth and development. A principal engineer's role in providing feedback extends beyond correcting mistakes — it's about guiding junior engineers in refining both their technical skills and professional behaviors. Whether you're offering insights on specific artifacts like design documents, RCAs, code reviews, or addressing broader behavioral patterns, your feedback should aim to build confidence, reinforce positive actions, and guide improvement. By carefully balancing feedback on both *what* they create and *how* they operate,

**Antipattern**

When giving feedback, especially in code reviews, avoid rewriting a junior engineer's work in your own voice. It may feel efficient in the moment — but it short-circuits the learning process and can erode confidence. Instead, guide them toward better solutions by asking questions or explaining trade-offs. Help them understand *why* a change might be beneficial rather than *what* the exact fix should be. Good feedback builds skill, not dependence.

you not only help junior engineers grow but also strengthen the working relationships within your team.

**Key principles for effective feedback**

1. **Be specific and actionable.** General feedback like "good job" or "you need to improve" doesn't provide enough direction. Instead, focus on specific actions or behaviors and how they can be improved or replicated.
2. **Balance positive and constructive feedback.** Start by recognizing what went well to build trust and reinforce strengths. Then offer clear, constructive feedback. Aim to end with encouragement or a forward-looking note — something that helps them stay motivated and focused on growth.
3. **Focus on outcomes, not the person.** Ensure that feedback is directed at the actions or outcomes rather than the individual's character or personality. This helps maintain a professional tone and ensures that feedback is received as constructive, not personal.
4. **Encourage two-way dialogue.** Feedback should be a conversation, not a monologue. Invite junior engineers to share their perspectives, ask questions, and express any concerns. This exchange can provide additional context and helps them feel more engaged in their own development process. Don't be afraid to explicitly ask "How can I help you?" or "What can I do to support you?"
5. **Be empathetic.** Understand that receiving feedback can be difficult, especially for someone early in their career. Approach the conversation with empathy, recognizing that your goal is to support their growth, not just critique their work. Acknowledge their reasoning before addressing gaps in the work, reinforcing their value while guiding improvement.

## **Technique:** Involve juniors in decision-making

Involving junior engineers in decision-making processes is a powerful way to accelerate their growth, boost their confidence, and foster a sense of ownership and accountability. When junior engineers are included in decisions, they gain valuable insights into the complexities of engineering projects, from technical trade-offs to stakeholder considerations. This involvement not only helps them develop critical thinking and problem-solving skills but also builds trust when they feel their contributions are valued and their perspectives respected.

Furthermore, their fresh perspectives can lead to innovative solutions that more experienced team members might miss. Trusting them in this process also builds their confidence, preparing them for greater responsibilities and future roles. Involving juniors fosters a collaborative culture, reinforcing the idea that every team member has something valuable to contribute.

**Tips for involving juniors in decisions**

1. **Invite input during planning.** During project planning and design phases, actively solicit input from junior engineers. Ask for their opinions on technical approaches, potential risks, and project timelines. Their point of view can be invaluable in identifying potential issues early on.
2. **Assign decision-making roles.** Give junior engineers specific roles in decision-making processes, such as leading a task force on a particular issue or conducting research to inform a technical decision. This not only provides them with hands-on experience but also highlights their importance to the team.
3. **Mentorship during decision-making:** Pair junior engineers with more experienced team members during decision-making processes. This mentorship allows them to observe how decisions are made and gradually take on more responsibility as their confidence and skills grow.
4. **Feedback on contributions:** After involving junior engineers in decisions, provide feedback on their contributions. Discuss what worked well and areas where they could improve. This reinforces the learning experience and shows that their input is valued.

5. **Iterative involvement:** Start by involving junior engineers in smaller, lower-stakes decisions and gradually increase their involvement as they gain more experience and confidence. This approach helps them build decision-making skills progressively.

**Scenario**

You're leading a large-scale API integration project that impacts both Zillow.com and the Zillow mobile app. The team is faced with a critical architectural decision: whether to refactor a legacy system or build a new service from scratch. While senior engineers are discussing the trade-offs, you recognize an opportunity to involve Sarah, a junior engineer who recently joined the team.

You approach Sarah and ask for her input on the decision. She hesitates, unsure if her perspective will add value, but you assure her that fresh insights are often invaluable. You suggest she lead a small research task, analyzing the technical and business impacts of both options. Before she starts, you offer her guidance on how to structure her findings and encourage her to think about not just technical feasibility but also stakeholder impact and long-term scalability.

Sarah spends the next few days gathering data, researching similar projects, and discussing potential risks with other engineers. After compiling her analysis, she's ready to present her findings. Before the team meeting, you coach her on how to communicate her ideas concretely, focusing on measurable outcomes like system performance and developer efficiency. During the meeting, you support her by asking clarifying questions and guiding the discussion, leading the team to lean toward creating a new service based on her recommendations.

This process boosts Sarah's confidence, and she feels more engaged with the team. More importantly, she trusts you more deeply, knowing you value her input and are invested in her growth. As the project progresses, Sarah identifies a potential issue in her part of the implementation—an early assumption in the design isn't holding up under real-world conditions. Rather than second-guessing herself or hesitating, she proactively reaches out to you, clearly articulating the problem.

Because of the trust and relationship built earlier, Sarah feels comfortable flagging the issue early. Together, you brainstorm solutions and involve the team, adjusting the design before the problem grows worse. Had Sarah not reached out, the flaw

would likely have gone unnoticed until much later — possibly post-launch — creating costly delays and impacting user experience.

By involving Sarah early in the decision-making process and creating a strong relationship, you've not only accelerated her growth but also improved the project's overall outcome. The trust built through this mentorship results in a proactive, collaborative approach that prevents costly mistakes and strengthens the team's success.

| Exercise | Think about your relationships with the junior engineers in your org. Are you helping shape them into better engineers? Have you used any of the techniques listed? How successful have you been? |
| --- | --- |

Section 2

# Building relationships with management

> *"You use data to bring clarity to contentious issues."* Communication facet
>
> *"You are expected to identify, define, socialize, and break down novel problems in your organizational domain."* Strategy facet

*— Zillow Leveling Guide*

As a principal engineer, your relationship with your manager evolves into a partnership where both of you act as force multipliers for the team. While you proactively drive technical improvements through architecture, coding, and managing technical debt, your manager focuses on enhancing individual performance, communication, and collaboration. Together, you should complement each other's strengths to create a high-performing organization.

## **Technique:** Managing up

Managing up is key for principal engineers to build strong relationships with leadership while ensuring their work aligns with broader organizational goals. To do this effectively, start by understanding your manager's priorities and how your efforts contribute to them. In your 1:1s, ask about key objectives and adjust your approach to ensure your work supports the bigger picture.

When raising concerns, don't just present problems — offer solutions. This proactive mindset positions you as a problem solver. Similarly, keep your communication clear and concise. Focus on the outcomes that matter to your manager, avoiding unnecessary technical details unless requested.

Seek feedback regularly, not just on technical tasks but on your overall contribution to organizational goals. Adapting your communication style to suit your manager's preferences is also important — some prefer detailed reports, while others value high-level summaries.

Managing up also means driving strategic initiatives that align with company objectives. Present these initiatives in a way that demonstrates their impact on both technical and business goals. Sometimes you'll need to present potential risks by escalating up. Escalate issues thoughtfully by highlighting risks and also proposing solutions. Show that you're focused on resolving challenges and not just complaining about them.

In short, managing up involves aligning with leadership's goals, communicating effectively, and proactively addressing concerns — ensuring both your success and that of the organization.

## **Technique:** Getting the most out of a 1:1

One-on-ones with your manager are a powerful tool for building the relationship. At the principal level, the dynamic shifts into a collaborative partnership, where you're expected to take initiative and guide the direction more in these meetings. This shift reflects the increased autonomy and responsibility that comes with a principal role, where you're not just executing tasks but driving strategy and influencing broader outcomes.

In your 1:1s, the focus will likely move away from straightforward action items, as those can be handled efficiently over Slack. Instead, your 1:1s can center on higher-level strategizing, providing feedback on team dynamics, and discussing the broader impact of your work. This is an opportunity to align on vision, address any potential roadblocks, and ensure that both you and your manager are on the same page regarding your role in driving the team's technical direction.

Remember, while the fundamentals of a good 1:1 — like preparation and clear communication — still apply, the expectations at the principal level are heightened. Your ability to run these meetings effectively will be a key factor in how well you can leverage your position to influence and shape the direction of your team and organization.

**Scenario**
You've noticed a team in your org is consistently accumulating technical debt and slowing down feature delivery. In your next 1:1 with your manager, you prepare to discuss the issue by focusing on how it affects your company's goals. You start

the conversation by acknowledging the push for faster releases and business growth, then highlight how the technical debt threatens both.

Instead of simply raising the problem, you diagnose that the team's struggles are caused by a poorly-organized code base. The team recognizes this, but has struggled to prioritize work to refactor. After collaborating with other engineers, you propose a phased refactoring plan that balances reducing debt with ongoing feature work. You explain the risks of inaction but emphasize that your solution will mitigate those risks and ensure long-term system stability.

When your manager expresses concerns about potential delays, you share a roadmap that maintains momentum on product development while addressing the underlying technical issues. You focus on delivering outcomes that align with the organization’s priorities and ask for feedback to ensure your approach fits within broader objectives.

By the end of the conversation, your manager is on board with your plan, recognizing your proactive problem-solving and strategic thinking. This strengthens your relationship and demonstrates your leadership in addressing both technical and business concerns.

| Exercise | Write down three key decisions or strategies that you’ve discussed in recent 1:1s with your manager. Identify the outcomes expected for these decisions. Are you and your manager aligned? |
|---|---|

## **Technique:** Status updates and transparent communication

Your status updates and transparent communication are vital components of your effectiveness as a principal engineer, especially when working closely with management and leadership. By regularly and clearly communicating, you ensure that everyone is aligned on your progress, challenges, and next steps. This not only builds trust but also fosters accountability and enables you to proactively solve problems before they escalate. When you communicate transparently, you maintain your credibility and contribute to a culture of openness, where both successes and setbacks are shared honestly. This approach not only keeps your projects on track but also strengthens your relationship with leadership, demonstrating your commitment to the team's and organization's success.

**Antipattern**

Avoid providing unfounded answers. It can be tempting, especially under pressure, to give a response rather than admit uncertainty. This approach can lead to misinformation and erode trust. Be clear when you don't know something, and outline the steps you'll take to find the correct answer.

Best practices for status updates include:

1. **Be concise and focused.** Keep your status updates concise and focused on key points. Highlight the most important developments, including progress made, current challenges, and any decisions that need to be made. Avoid overwhelming leadership with too much detail; instead, provide a high-level overview with the option to dive deeper if needed.
2. **Use data to support updates.** Whenever possible, use data to back up your status updates. Metrics, timelines, and other quantifiable information make your communication more credible and help leadership quickly grasp the current state of the project. Visual aids like charts or dashboards can also be effective in conveying complex information succinctly.
3. **Highlight risks and challenges.** Transparent communication means not only sharing successes but also being upfront about risks and challenges. If a project is behind schedule or facing a significant obstacle, address it openly and provide a plan for mitigation. This approach shows that you are proactive and committed to finding solutions, rather than avoiding difficult conversations.
4. **Be honest about uncertainty.** In complex projects, there will inevitably be unknowns. If there are areas where you lack complete information or where outcomes are uncertain, be honest about it. Transparency about uncertainty allows leadership to understand the situation better and provides an opportunity to discuss contingency plans or additional resources.
5. **Document and share.** After providing a status update, document the key points and action items, and share them with the relevant stakeholders. This practice reinforces the message, ensures everyone is on the same page, and creates a record that can be referred to later.
6. **Use AI to streamline writing.** Large language models (LLMs) can help you compose or refine status updates more efficiently. They're especially useful

for summarizing project details, tightening language, or brainstorming phrasing.

### Section 3

# Building relationships with project stakeholders

> *"You develop partnerships with key stakeholders, seeking to influence designs across team boundaries and proactively inform product roadmaps."* Strategy facet
>
> *"You participate in the planning rhythms of your organization. You are an expert in your business domain, shaping technical investments to unlock long-term business capabilities."* Strategy facet
>
> — *Zillow Leveling Guide*

Successful software projects require the alignment of technical goals with the needs of various stakeholders. Building strong relationships with product managers and other non-technical stakeholders ensures that the engineering team is delivering value that meets the broader organizational goals. These relationships are critical for negotiating priorities, managing expectations, and driving cross-functional design that integrates diverse perspectives.

## **Technique:** Laying the groundwork with *nemawashi*

*Nemawashi* is a Japanese term that refers to the informal process of laying the groundwork for a proposal or decision by engaging key stakeholders and getting their buy-in before formal discussions begin. You might also hear this referred to as "the meeting before the meeting". It's a powerful technique that principal engineers can use to build relationships with project stakeholders. Here's how:

1. **Early engagement:** Have informal, one-on-one conversations with stakeholders to understand their perspectives and address concerns before formal discussions begin.
2. **Tailor proposals:** Use insights from early discussions to tailor your proposal, ensuring it aligns with stakeholders' concerns and suggestions.

3. **Gradual consensus building:** Build consensus gradually through small, informal discussions, making the formal approval process smoother.
4. **Reduce resistance:** Address potential objections early to minimize resistance and ensure a smoother decision-making process.
5. **Strengthen relationships:** Demonstrate respect for stakeholders by involving them early, fostering a collaborative environment, and building stronger relationships.

### Scenario

You're the principal engineer for a Zillow platform team that has come up with an idea for a unified logging service to simplify troubleshooting across microservices. Instead of jumping straight into building, you decide to engage key stakeholders early to gather feedback and ensure alignment. This *nemawashi* approach will help strengthen your proposal and build support before the formal review.

You start by having informal conversations with the SRE team, where you learn about their struggles with inconsistent logging formats. Next, you reach out to a principal engineer from a team that heavily uses the existing logging system. Known for being vocally critical of new initiatives, he challenges the proposal, raising concerns about performance degradation and the potential impact on his team's workflows. You engage in the tough conversation, digging into his pain points. Rather than avoiding the pushback, you recognize this as an opportunity to address critical concerns early. His feedback leads you to think more carefully about system performance and backward compatibility, prompting adjustments to the design.Following that conversation, you meet with a product manager, who highlights the importance of a smooth transition plan to avoid disruptions to current workflows. This reinforces the need for a seamless migration strategy, and you incorporate a phased rollout approach to ensure minimal disruption.

When you meet with a director of engineering, he expresses skepticism, having seen similar projects fail due to misalignment. His feedback motivates you to clearly define the value proposition and include metrics for success in your proposal. Finally, the principal security engineer raises concerns about privacy, prompting you to integrate security safeguards into the design early on.

Armed with this feedback, you refine the proposal to address each stakeholder's concerns. By the time the formal review happens, discussions flow smoothly — each stakeholder feels heard, and their needs are addressed. This leads to a

**Antipattern**

Focusing too much on pushing your own agenda without genuinely listening to the needs and concerns of stakeholders can erode communication and trust, as stakeholders may feel ignored. To avoid this, actively engage, ask thoughtful questions, and understand their concerns. By seeking alignment and balancing both your goals and theirs, you create a more collaborative environment, ultimately driving better outcomes and building stronger relationships.

strong buy-in across the board, ensuring the project can move forward without resistance.

Through this process, you not only improved the design of the logging service but also built trust with your stakeholders. By embracing difficult conversations and using *nemawashi*, you laid the groundwork for both a successful project and stronger relationships within the organization.

## **Technique:** Collaborate with PM to understand stakeholder priorities

Working closely with a product manager is critical for aligning technical decisions with business goals and ensuring project success. While product managers often lead stakeholder engagement, it is equally important for Principal Engineers to have a clear understanding of stakeholder priorities to help drive coherent technical decisions. Together, this partnership ensures that the project meets or exceeds stakeholder expectations and helps navigate the complexities of project execution.

At its core, an organization thrives on the partnership between Product and Engineering. Sometimes, this relationship can be difficult to navigate. As engineers, it's natural for us to leap immediately to focus on the problems that need to be solved — which is easy for other stakeholders to interpret as focusing on why something *can't* be done. Feasibility is important, of course, but we must be mindful to build a constructive relationship.

Try to focus on what's possible given technical constraints. Offer possibilities, and do so as early in the process as possible. This helps educate others as to the full range of the trade-offs and strategic options that they can consider. Your partners will perceive that you are sharing information and offering solutions.

This approach — shifting from feasibility concerns to solution-oriented discussions — positions you as a key player within the organization. It builds trust, secures stakeholder buy-in, and demonstrates that you are not just a technical lead but also a strategic partner, actively contributing to the project's success. By encouraging this type of collaboration, you enhance your influence and help deliver solutions that drive both technical and business success.

**Tips for collaborating with project managers**

1. **Strengthen your partnership.** Use recurring 1:1s as a time to align on priorities, share insights into technical constraints, and understand the broader business context. Regularly discussing project goals, challenges, and potential trade-offs builds a strong partnership.
2. **Collaborate on stakeholder interviews.** Use this opportunity not only to align technical strategies with business objectives but also to dig into the underlying "why" behind their needs. By asking the right questions, you can address root causes and contribute more effectively to the project's success.
3. **Align on key objectives.** Help identify and agree on the key objective. Ensure that both the technical and business aspects of the project are aligned with stakeholder goals. Sync regularly to ensure that your understanding of these objectives is up-to-date and accurate.
4. **Clarify success metrics.** Work together to clarify the success metrics stakeholders will use to evaluate the project. While the product manager may focus on business-related metrics, you can ensure that technical performance metrics are also considered.
5. **Communicate trade-offs.** When trade-offs are necessary, you can communicate together, to present a unified explanation of why certain decisions are being made and how they impact different stakeholders. This coordinated communication builds trust and ensures that stakeholders understand the reasoning behind both business and technical decisions.

| Exercise | Think about your relationship with key stakeholders you work with. How would using the techniques listed above help you in your latest proposal? |
|---|---|

Section 4

# Building your network

> *"While you are an expert in your craft, you continue to learn and grow. You relentlessly seek to better understand the particulars of your dependencies, your customers, and how to influence others across your organization/the company."* Learning facet
>
> *"You share lessons with the larger engineering community."* Operations facet

*— Zillow Leveling Guide*

While team autonomy can drive ownership and efficiency, other times it can create silos that lead to misalignment, duplicated efforts, and missed opportunities. Building relationships across organizations helps break down these silos, fostering cross-team collaboration and leading to more coherent and scalable solutions. However, networking shouldn't be limited to just engineers — it's equally important to build relationships with key stakeholders such as product managers, business leaders, and others who can amplify your influence and impact. These connections open up channels for sharing best practices, learning from diverse perspectives, and identifying cross-functional opportunities for innovation that span multiple domains. Engaging with Product, in particular, ensures that your technical decisions are aligned with the broader business strategy, driving both technical and organizational success.

## **Technique:** Engage in cross-orginizational initiatives

Building your network outside of your immediate team is essential for broadening your influence and understanding within the company. One of the most effective ways to achieve this is by proactively participating in cross-orginizational initiatives. This could include projects related to company-wide process improvements, new product development, or strategic initiatives that require input from multiple teams. These initiatives provide opportunities to collaborate with colleagues from different groups, share knowledge, and gain insights into the broader organizational goals and challenges. Additionally, participating in these initiatives allows you to build relationships with colleagues in other areas of the

company. These relationships can be invaluable when seeking support for your projects or when looking for collaborators for future initiatives.

**Strategies for participating in cross-organizational initiatives**

1. **Seek out opportunities.** Seek out opportunities to participate in cross-organizational initiatives. Express your interest to leadership and colleagues, and be open to contributing wherever your skills are needed.
2. **Leverage your expertise.** When participating in these initiatives, bring your technical expertise and leadership skills to the table. Be prepared to offer insights and solutions that align with the goals of the initiative while also demonstrating your ability to collaborate effectively with others from different backgrounds.
3. **Understand organizational priorities.** The importance of understanding priorities was highlighted earlier, and it's even more crucial when stepping outside your team to offer assistance. Before diving into a cross-organizational initiative, take the time to understand the broader organizational priorities and how the initiative aligns with them.
4. **Follow up and stay connected.** After the initiative concludes, maintain the relationships you've built. Follow up with your new contacts, share insights from the project with your own team, and look for future opportunities to collaborate. Staying connected helps solidify your expanded network and keeps you top of mind for future cross-organizational work.
5. **Engage through innovation events.** Innovation-focused events — like Hack Weeks (Hackathons), Innovation Weeks, or Ship It Days — are great opportunities to collaborate across team boundaries. They allow you to prototype ideas, explore new technologies, and work with people outside your immediate group. These moments often spark deeper cross-org relationships and can lead to impactful follow-up work.

## **Technique:** Make connections through forums

Internal forums, like dedicated channels on Slack or recurring lunch-and-learn meetings, are powerful tools for networking, knowledge sharing, and problem-solving among engineers. As a principal, you should look to actively

leverage and engage in forums that exist today to grow your network around the company.

Start by **exploring and engaging** in these forums. For example, many companies have dedicated spaces (at Zillow, we have the **#staff_eng** Slack channel) for principal+ engineers to connect, share insights, and discuss challenges specific to senior engineering roles. Engage there and in other forums by contributing to discussions, asking questions, and sharing your expertise. Share interesting articles, pose thought-provoking questions, and join in discussions on others' contributions.

This increases your visibility to others, making them more likely to reach out to *you* to ask questions or engage in discussion. If you welcome and engage warmly with others when they seek you out, your network will organically grow.

By the same token, don't hesitate to **reach out to others** in these forums who appear to share the same interests or are working on similar problems. A direct message on Slack is a great way to introduce yourself, and asking to grab a half-hour to make introductions over a video call is an excellent way to make a human connection. The forum is a place to find others; how you follow up determines whether you'll nurture a more meaningful relationship.

## **Technique:** Leverage relationships to expand your network

Your existing relationships within the organization are valuable assets that can be leveraged to build and expand your network. By using your existing connections, you can efficiently connect with colleagues across different teams or departments, broadening your influence and exposure within the organization. This approach not only helps you gain access to a wider pool of resources, including knowledge, expertise, and potential collaborators, but also strengthens your influence, making you a more integral part of the organization's engineering staff.

**Tips to leverage relationships to expand your network**

- **Identify strategic connections**. Start by identifying who in your current network could introduce you to others who might be valuable contacts. Think about the goals you want to achieve — whether it's finding someone

**Antipattern**

Focusing solely on self-promotion rather than genuine relationship building can lead to superficial relationships that lack trust and reciprocity, ultimately limiting your ability to effectively leverage your network. By focusing too much on self-promotion, you may miss opportunities to collaborate, help, and build long-term alliances.

with a specific skill set, learning more about a different department, or gaining insights into another area of the business. Reach out to those in your network who are well-positioned to make these introductions.

- **Build on shared interests or goals**. Use your existing relationships to identify shared interests or goals between yourself and potential new contacts. For example, if you and a colleague are both passionate about a particular technology or approach, they might know others in the organization who share that passion and could introduce you to them.

- **Follow through and build new relationships**. Once an introduction is made, it's important to follow through by actively engaging with the new contact. Take the time to understand their challenges, offer your help, schedule a 1:1, and find ways to collaborate. Building a relationship requires effort, so invest in nurturing these new connections just as you would with your existing ones.

**Exercise**

Let's continue to build your network. Think about 3-5 people you would like to meet. Write those down, and reach out to them directly or ask someone in your network to make the introductions.

**Part III**

# Leveraging influence

Influence is the ability to intentionally shape — without direct authority — the behavior or attitudes of others. Influence is carefully accrued over time by building trust, demonstrating expertise, and effectively communicating ideas to inspire and guide. The ability to influence others is central to your long-term ability to create impact. It doesn't matter how good your ideas are if you cannot influence others to embrace those ideas and act upon them.

Influence differs from persuasion, which is more tactical and direct. Both persuasion and influence are essential skills for a principal engineer, but influence is more broadly impactful and enduring. Unlike persuasion, influence relies on long-term relationships and a consistent demonstration of value, and can sometimes be more subtle. It takes time and great care to cultivate the seeds of influence, and it can be frustratingly slow before you begin to see the fruits of your labor, but the results are often more substantial and lasting.

## Influencing down, up, out, and across

Principal engineers are further removed from project execution than junior engineers and lack the direct reports of a manager, but they're expected to drive broad, lasting impact. That's where influence enters the picture. By working through others, principal engineers can drive outcomes far beyond what they could achieve on their own.

Earlier we discussed how to grow and strengthen your influence by building relationships. Now we'll cover how to effectively leverage your influence to create change. The first section will discuss how to drive cultural change, and the second section will explore techniques for driving change on large technical projects. You can think of the first section as influencing within your organization and the second section as influencing outside it, but the techniques in each section can often be applied more broadly.

**Section 1**

# Cultivating a standard of excellence

> *"To your team(s), you set a tenor of excellence: you are skilled at upholding a high-quality bar without alienating others, seeking to cultivate and mentor."* Leadership facet
>
> *"You are the engineering face for your team(s): Principal Engineers, Product Managers, and others all seek you out and trust you to represent the technical capabilities and constraints of your space."* Leadership facet
>
> *"You understand that your impact scales with your influence: you cannot uphold these standards solo, but instead must rely on the team to embody them even when you're not looking."* Leadership facet
>
> — *Zillow Leveling Guide*

Driving cultural change within an organization requires intentional effort and strategic direction. This section explores practical techniques for creating and sustaining an engineering culture that prioritizes quality, innovation, and continuous improvement.

## **Technique:** Lead by example

As a principal engineer, your actions and behaviors influence the behaviors of those around you. By virtue of your title, you are a role model to junior engineers. Your title signifies that your company has entrusted you with leadership responsibilities, and others will observe your actions accordingly. What you do and how you do it will set the standard for others to emulate, and they in turn are likely to model those behaviors to their colleagues, creating a ripple effect that can change the overall engineering culture. If you're intentional about your goals, your words and actions become a powerful tool to influence others. For example, by showing dedication to quality and teamwork, you foster a culture where high standards are the norm.

By the same token, setting a poor example can be contagious as well. If you neglect operational excellence, those around you are likely to do the same. If you don't prioritize thorough design documentation, it will be challenging to persuade

others to invest effort in their designs. If you are combative and quick to criticize, you'll struggle to foster collaboration. Ineffectiveness can diminish your own credibility, causing others to take your words and actions less seriously. To effectively influence others, you must first set a good example.

**Key practices when leading by example**

- **Lead with conviction**. When you act decisively, even in the face of ambiguity, you show others how to navigate uncertainty with purpose. Your decisiveness minimizes unnecessary delays and fosters a culture of progress and purpose.
- **Be open and encourage feedback**. By actively seeking and valuing feedback, you create an environment where learning and continuous improvement are prioritized. This openness creates the psychological safety for others to voice concerns, suggest ideas, and engage in constructive disagreements — which in turn leads to better outcomes and a more collaborative team culture.
- **Listen and ask questions.** Active listening and curiosity shows that you value the perspectives of others. This approach not only promotes openness and transparency but also helps to surface hidden issues or innovative ideas.
- **Bias toward action**. Show a proactive attitude by taking initiative and addressing issues head-on. Similar to leading with conviction, this approach promotes efficient problem solving and encourages your colleagues to adopt a similar mindset.
- **Push back on incomplete requirements and poor assumptions**. By challenging incomplete or flawed assumptions — and encouraging others to do the same — you mitigate risk and promote more vigorous design thinking.
- **Practice articulate and timely communication**. Helps prevent misunderstanding, keeps projects on track, and fosters a more collaborative work environment.

Leading by example doesn't mean you must be perfect! Owning mistakes can be a powerful example as well. It encourages others to take ownership of their own actions and helps foster an environment where it's safe to take risks and ask for

help. Acknowledging your own weaknesses builds trust and demonstrates that leadership is about progress, not flawlessness.

**Scenario**

You're the principal engineer responsible for several systems that support the search experience. It's Friday afternoon and one of your teams is under pressure to deliver a new feature before the weekend. Alerts suddenly indicate the search experience is intermittently failing, frustrating users. Recognizing the urgency, you see this as a critical moment to guide the team through the resolution process and to lead by example.

You reach out to the junior engineer who is on call and has been investigating the issue. The engineer is unsure whether a recent code change might be the root cause of the issue and suggests rolling back as a potential fix. However, the engineering manager, reluctant to miss the end-of-week deadline, suggests investigating further before making any decisions.

Sensing the junior developer's uncertainty, you engage with her directly, asking for her thoughts on the situation. She carefully outlines the pros and cons of rolling back versus continuing to investigate. Acknowledging both sides, you emphasize the importance of system stability and user experience, and suggest bias toward rollback as a best practice. You explain that although further investigation might pinpoint the exact issue, the risk of ongoing user frustration and potential escalation outweighs the benefit of a potential quick fix. Your reasoning resonates with the junior developer, who feels valued and supported in her decision-making process.

With your support, the junior developer presents the rollback recommendation again. Initially the manager resists, concerned about missing the deadline. However when you stand firmly behind the junior's assessment, the manager relents, recognizing the need for caution and the value of your experience.

While the developer initiates the rollback, you keep stakeholders informed of progress and answer questions in the incident Slack channel. This approach not only reinforces the importance of clear communication but also frees the junior developer to focus on mitigating the issue. The roll back restores the search systems to a stable state and users are able to search for homes again.

**Antipattern**

Building a platform can be a powerful way to enable teams and drive consistency — but it's also a major investment of time and resources. It shouldn't be pursued by default or for its own sake. Often, the goals of a "paved road" can be met more efficiently through simpler, lighter-weight solutions, such as well-documented templates, reusable libraries, or clear best practices. Before deciding to build a platform, carefully consider whether the problem truly calls for one, and whether your proposed solution will be broadly adopted and maintained over time.

On Monday morning, the team gathers to write the root cause analysis (RCA). The junior developer identifies the code defect and writes a fix. You review the preliminary RCA and the code change. You provide feedback, focusing on clarity and completeness, before giving the green light to release the feature the next day.

Throughout this process, you've demonstrated the importance of decisive action balanced with thoughtful consideration. By supporting your team and making the right call under pressure, you not only resolved a critical issue but also modeled a leadership style rooted in care, conviction, and collaboration. This experience reinforces your role as a trusted leader and helps build a culture where both urgency and quality are held in high regard.

## **Technique:** Pave the way

A principal engineer's influence extends beyond direct interactions and decisions. It permeates through the tools, processes, and standards you help establish, which will guide and shape the work of others, even in your absence. Two powerful concepts that enable this kind of influence are "paved roads" and "guardrails."

*Paved roads* refer to the preferred practices or tools that are well-supported, easy to follow, and designed to encourage best practices. A paved road might be a tool for testing, deployment, or monitoring. It could be a template for technical documentation, architectural patterns, or coding standards. When you create paved roads, you guide others toward effective solutions, reducing friction and avoiding common pitfalls. By making the right thing easy to do, you influence others' choices and behaviors, driving consistency and quality across the organization.

The hallmarks of a paved road are:

- Self-service tools or well-documented guidance
- Ease of adoption
- Enforcement of best practices for common use cases
- Optional, allowing the developer to go “off-road” when needed

A paved road can be especially powerful when paired with the closely-related concept of *guardrails*, constraints or procedural hindrances that ensure engineers don’t veer off course. They establish boundaries within which engineers have the freedom to operate, while still maintaining alignment with accepted standards. Guardrails might include automated testing, security policies, or linters. Guardrails can also be procedural: merge request policies, design approval processes, or documentation standards. Guardrails prevent costly mistakes, technical debt, or divergence from best practices by clearly defining **what should not be done.**

The hallmarks of guardrails are:

- Enforced standards
- Simple, not overly restrictive
- Define boundaries of acceptable practices
- Provide flexibility within constraints

While a paved road makes it easy for an engineer to do the right thing, guardrails make it harder for them to do the wrong thing. Paved roads offer guidance on the *how*, guardrails enforce the *what not*. Together they create a framework that influences others’ actions and decisions, enabling you to steer the technical direction of your organization even when you’re not directly involved. Look to create paved roads and guardrails for your engineers as these can be an effective and enduring way to cultivate high standards of engineering excellence.

**Scenario**
Following the resolution of the search functionality issue, you recognize the need to prevent similar problems in the future. To “pave the way” for smoother development and operations, you look for ways to implement paved roads and guardrails within the existing engineering processes.

When your system was down, aggressive retries by one of the dependent systems had caused a cascading failure. The retries themselves had overwhelmed your system and delayed recovery. To create a paved road, you propose the adoption of an open-source HTTP client that uses a circuit breaker pattern with exponential backoff to provide more resilience. You work closely with the junior engineer who was on-call during the incident, to integrate the new HTTP client library into the dependent system. Together, you craft clear documentation explaining the library's value and provide example use cases to accelerate its adoption. It quickly becomes the standard client library for your teams, ingraining resilience into the day-to-day development process of your team.

Next, you introduce guardrails by identifying that static code analysis could have spotted the code defect before it reached production. You introduce a linter in the CI/CD pipeline to flag potential issues in the future and hold training sessions for the teams to understand how to use it effectively. By implementing automated safeguards, you've helped the teams work more effectively, reduced the risk of errors, and promoted a culture of excellence. Your proactive approach ensures that the system remains robust and that engineers can focus more of their time on meeting business objectives, rather than firefighting production issues.

## **Technique:** Amplify success

When thinking about advancing cultural change, it's easy to focus on the gaps where we're falling short of the technical excellence we expect of our organization, but one of the most effective strategies for driving change as a principal engineer is to amplify success. Identifying and leveraging existing successes can spread their impact, build momentum, and influence broader change.

Amplifying success is about recognizing where something is working well and strategically expanding its influence. Whether you're spotlighting a successful project within your organization, drawing from best practices in another org, or looking outside the company for inspiration, the key is to harness that success as a catalyst for broader change. This approach not only spreads effective practices but also creates a positive feedback loop, where each new success paves the way for the next, further strengthening engineering culture.

Although there are no rigid set of steps for amplifying success, the general approach might look something like this:

1. **Identify success stories**. Look for instances where teams or individuals are excelling, whether in productivity, innovation, quality, or any other relevant dimension. Success can be found internally or externally — don't limit yourself to only what's been done within your organization. Building a strong professional network helps you identify these success stories more easily.

2. **Understand success drivers**. Once you've identified a success, dig deeper to understand what's driving it. This could involve direct conversations, gathering feedback, or informal observations. Look for actionable insights — key practices, tools, or processes — that others could replicate.

3. **Develop a plan**. With a clear understanding of what made the success possible, create a plan to share and scale those practices. Tailor your approach to the needs of your audience. This might involve creating workshops, writing guides, or organizing presentations. The goal is to make it easy for others to adopt these successful practices.

4. **Recruit champions**. Invite others to help lead the cultural change. Whenever possible enlist the help of those who were directly involved in the original success. Their firsthand experience lends them credibility and makes them effective advocates for spreading the practice. If the advocates are from your company, it may be a valuable leadership opportunity for them, and you can provide coaching as needed. You can also step into the champion role yourself, demonstrating your commitment and conviction — especially when the success comes from individuals or teams outside your company.

5. **Share, monitor, and support**. Now it's time to execute on the plan. As you roll it out, keep a close eye on its impact. Encourage feedback, celebrate wins, and be prepared to adjust your approach as needed. Driving cultural change isn't a one-time event — it's an ongoing process that requires nurturing and support to truly take root.

### Scenario

After your teams adopt the HTTP client library, you notice significant improvements in system stability and performance. Recognizing the potential for

broader impact, you set out to amplify this success by promoting the use of the library to the broader organization.

Working with the junior engineer again, you start by gathering metrics to demonstrate the positive impact of the new HTTP client library: reduced downtime, fewer cascading failures, and improved incident recovery times. You also collect feedback from developers on its ease of use. You and the junior engineer create a presentation for an upcoming all-hands engineering meeting of your org. Together, you share the metrics and tell the success story, clearly articulating the value the library has brought to your team's systems. You encourage other teams in the organization to adopt the library to achieve greater system resilience, and offer to be a resource to engineers who have questions.

Over time, more teams adopt the library, leading to widespread improvements in system stability and performance. By amplifying the initial success, you've had a wider impact and you've also fostered an engineering culture that learns and grows from previous mistakes. You've celebrated a collective victory, recognized the contributions of a colleague, and given that colleague an opportunity for greater visibility and leadership that can continue to provide value in the future.

## **Technique:** Build communities of technical excellence

Engineering rituals can be powerful tools for driving and sustaining cultural change. They help you keep a pulse on your organization and better represent its interests to others, and they allow you to influence and shape the engineering culture within your organization. These rituals do more than just uphold quality — they create habits that become ingrained in the engineering ethos, ensuring high standards are maintained even when you're not directly involved. As these practices take root, they build momentum, creating a virtuous cycle of self-improvement. Over time, this cultivates a culture of constant learning and empowers others to take ownership of these practices themselves, ensuring that the culture endures.

Some common engineering rituals:

- **Design reviews** - Structured sessions where engineers present their design proposals to peers for feedback. By bringing together diverse perspectives, design reviews expose engineers to different patterns and approaches, helping them spot issues or improvements they might have otherwise

**Antipattern**

Another pointless meeting: Be mindful that engineering rituals should provide value relative to the time they take. If a ritual is not delivering the desired impact, don't be afraid to change cadence, format, or even disband as needed. The focus should be on effectiveness.

missed. Effective for promoting architectural best practices, catching potential issues early, and staying up-to-date with the architectural landscape and capabilities.

- **Root cause analysis (RCA)** - Process for identifying the underlying reasons for defects or failures, and planning future mitigations. Useful to prevent recurring issues, improve operational rigor, and foster a culture of continuous improvement.
- **Project retrospectives** - Meetings held after project milestones to reflect on what went well and what could be improved. Drives team improvements, shares lessons learned, and enhances future project execution.
- **Operational metric reviews** - Regularly analyzing key performance indicators (KPIs) to assess system health and performance. Conducted to identify trends, improve observability, and promote operational excellence.
- **Enterprise architecture groups** - Collaboration of technical leaders to define and maintain technical strategy. Aligns technical decision-making with strategic initiatives and encourages cohesive system architecture.
- **Knowledge share sessions** - Informal sessions (e.g. lunch and learn, coffee chat, innovation hour) where peers share research, ideas, or experiences on various technical topics. Fosters continuous learning, innovation, and a collaborative culture.
- **Work-in-progress sharing** - Regular informal demos of ongoing work to peers and stakeholders, often at the team or organizational level. Effective for gathering feedback, promoting collaboration, and celebrating progress.
- **Guilds** - Communities of practice focused around a specific technical interest or skill, often in a way that cuts across organizational boundaries (e.g. DevOps, DataBricks, front-end development). Facilitates knowledge sharing, innovation, and development of best practices.

**Scenario**
You see an opportunity to enhance the engineering culture and amplify success stories by introducing a new monthly lunch-and-learn series focused on engineering best practices and operational excellence. For the inaugural session you plan a presentation focused on the incident that sparked the resiliency improvements. To empower and recognize emerging talent, you ask your junior colleague to lead the session and talk about being on call and playing a crucial role in the resilience improvements. You provide coaching and support in advance to ensure the first session is a success. The lunch-and-learn series continues, with engineers sharing their own lessons learned and ideas for improving engineering culture. These sessions not only provide valuable learning opportunities but foster a sense of community and collaboration across the organization.

You reflect back on the progress you've made and the impact you've had. By consistently and intentionally leveraging your influence, you've set into motion a cascade of improvements. Working through others, you've transformed a localized issue into a driving force for widespread cultural change, embedding principles of excellence, collaboration, and resilience across your organization.

| Exercise | Reflect on a time when you tried to influence a cultural change within your teams or organization. What techniques did you use? Any of these? What was challenging? What was most successful? |
|---|---|

## Section 2

# Driving impact through collaboration

| |
|---|
| *"You actively promote and model collaboration, listening to feedback from others and successfully mediating contentious technical discussions."* Leadership facet |
| *"You cultivate collaboration, listening to and integrating feedback across all levels."* Communication facet |

> *"You demonstrate effective verbal and written skills, communicating with everyone from the junior-most engineer to the senior leaders of your organization."* Communication facet
>
> *— Zillow Leveling Guide*

This section explores techniques for collaborative problem-solving when the technical challenge spans many areas of ownership and requires coordination across groups.

## **Technique:** Lead technical working groups

As a Principal Engineer, you'll often be tasked with solving technical problems that span teams or even organizations. When a technical challenge spans many areas of ownership, consider establishing a technical working group (TWG). A TWG is a temporary team formed to collaborate and rapidly address a particular technical challenge. The technical challenge could be designing a new product capability, migrating off a legacy technical system, or addressing a recurring source of developer friction.

When establishing a TWG, you'll want to clearly document the technical challenge and identify clear, measurable goals for success. Having a shared understanding of what success looks like is imperative to successfully addressing the problem.

Then you'll want to identify the right stakeholders to contribute to the solution. Having a wide professional network can accelerate finding the key contributors. You may need to meet with potential participants to identify who is best situated to address the problem and has the bandwidth to take on the commitments of the TWG. You don't need to include every expert — just those who know enough to identify when and whom to bring in for specialized input. You'll want a group that is large enough to represent the full scope of the problem being solved and typically no larger — having too many participants can encumber collaboration and slow decision making.

**Some tips for running a successful technical working group**

- Designate a directly responsible individual to lead the group. They should be skilled at meeting facilitation. This ensures that meetings are focused, decisions are made, and action items are followed through on.

- Set a regular meeting cadence to ensure teams are collaborating in a timely and effective manner.
- Set clear objectives before each meeting to keep the discussions focused and productive.
- Document decisions and action items. Follow up on these items in subsequent meetings to maintain accountability and progress.
- Establish standard communication tools and practices to minimize miscommunication and keep everyone informed. Some possibilities to consider: a project Slack channel, shared Google drive, and/or main Google document that links out to supporting materials.

**Scenario**
You're a principal engineer in the newly-minted Home Renovation organization. One of your first tasks is to help build a new user experience connecting homeowners with home renovation professionals, which will require coordination with several different engineering organizations. You work with product managers to gain insights into the business objectives and desired features for the contractor connection experience. With this knowledge, you meet with engineering managers and principal engineers to understand which teams need to be involved, their level of engagement in this new product offering, and to identify the best representatives for a technical working group.

Once the key stakeholders are identified, you work with your product partner(s) to document the goals and deliverables of the working group, following up when requirements are unclear. Together you document the objective: to create a seamless experience for homeowners to find, evaluate, and connect with reputable home renovation professionals. You set up an initial kickoff meeting to align everyone on the shared vision and ask teams to commit to an initial tech design and scoping, establishing a timeline and clear responsibilities for each team member.

As the working group progresses, a significant technical conflict arises. The existing monolithic architecture supports critical functionalities like user authentication, appointment scheduling, and search capabilities. Your teams suggest that to support the scalability and flexibility required for the new features it would be beneficial to break them into microservices, allowing them to scale independently and ensure long-term maintainability. Some of the backend teams

contend that leveraging the existing monolith would enable the project to ship more quickly, meeting the immediate market need with fewer initial disruptions.

Recognizing the need for a focused effort to resolve these issues, you decide to continue the discussion in a separate meeting to delve deeper into the technical trade-offs and to seek resolution — and the legwork you've put in to bring together the relevant stakeholders has laid the foundation for further collaboration and provided a forum where you can exert influence, even without direct authority. We'll pick up this scenario again after the next section which will address methods for resolving technical conflict.

## **Technique:** Resolve technical conflict

Conflicts can arise when collaborating across teams on a technical project. Others will often look to you as a principal engineer to resolve these conflicts in a positive and timely way. Propelling a project past stumbling blocks and friction can be some of the highest leverage activities you can undertake to have a broad impact. In this section we'll explore some practical techniques to address technical conflict, and some common situations where they can be helpful.

Even without direct authority, the principal engineer has a wide menu of techniques to choose from when resolving technical conflicts. Here are some common ones (and this list is by no means exhaustive):

**Understand the problem**

- **Ask why.** Get to the root of the business problem by asking: "What's the business value?" and "What problem are we trying to solve?" Similar to the five whys of a root cause analysis, keep asking until you get to the underlying needs and objectives.
- **Understand stakeholder motivations**. Be curious and ask questions about the perspectives and motivations of stakeholders. Try to appreciate the human aspects of the problem. In addition to better understanding the problem, this builds trust and opens up constructive dialogue.

**Create shared understanding**

- **Frame (or reframe) the problem**. Clearly document and define the problem, capturing the primary dimensions of the problem, possible resolutions, and their tradeoffs. This helps create a shared understanding and makes it easier to collectively problem-solve.
- **Decompose the problem**. Break a large, complex problem into smaller, manageable subproblems. Then prioritize the most critical aspects first. This makes the work less daunting and more achievable.
- **Use a proof of concept.** Discussed in detail in part one, developing a small-scale version of a solution can help test the feasibility of an idea without committing significant resources.

**Approach the problem differently**

- **Resequence the work**. Change the order of tasks in the project to reduce bottlenecks or dependencies. This can help streamline the workflow and improve efficiency.
- **Recruit an away team**. Temporarily assign engineers from another team to help with the work. This requires a well-defined scope of work and clear hand-offs when spinning up and spinning down the away team. Note however that adding people to a project may initially slow it down before it speeds up, and that not all work can be easily parallelized.

**Make a decision**

- **Persuade with data**. Use data to inform and support decision-making. This adds an objective perspective that can help resolve disagreement.
- **Bias toward action**. Encourage moving forward with decisions when faced with uncertainty. Taking action, even if imperfect, can often lead to further clarity, if not progress. This technique pairs well with seeking two-way doors, which was discussed in part one.
- **Escalate**. Present choices to leadership to differentiate between competing options. Be a good partner and use escalation, not to lay blame, but to gain clarity on prioritization and resource allocation.

The right technique for resolving technical conflict depends on the specific dynamics of the situation. As a principal engineer, you're expected to exercise

sound judgment in selecting the most appropriate tool to influence a positive resolution. This ability to discern and apply the right approach improves with experience and is a key marker of your leadership.

Below are some common situations in which technical conflict can arise and some possible approaches you could take to resolve them. You might use one or more of the techniques in combination.

*When there are two or more **competing technical approaches** and uncertainty about which one is best:*

- **Frame the problem**. Carefully document the approaches and their differences, highlighting tradeoffs between them. With this shared understanding, facilitate a discussion to arrive at the best course of action.
- **Use a proof of concept**. Create a proof of concept for one or more approaches, then leverage these to make a more informed decision about which is better.
- **Bias toward action**. Perhaps one of these approaches is a two-way door and it's better to make forward progress than deliberate in the hopes of gaining greater certainty.

*When one or more teams' **resource constraints** cause bottlenecks that slow down project progress:*

- **Understand stakeholder motivations**. Talk with resource-constrained teams to understand the source of the delays; the root problem may be one of these other common technical conflicts.
- **Resequence the work**. Can the work be ordered differently so that the resource-constrained team is not an immediate project dependency so the group can deliver on the business objectives more efficiently?
- **Recruit an away team**. Perhaps you or another team can offer assistance to accelerate the work by doing initial development and handing off afterwards.
- **Escalate**. Consider escalating to leadership to request additional resources or time.

*When two or more groups must collaborate on a body of work, but have **diverging priorities** that make that work high priority for one and low priority for the other:*

- **Ask why** Clarify the business value of the project and ensure all parties understand how the work contributes to the company's objectives.
- **Understand stakeholder motivations**. Work to understand the impact and importance of competing priorities, keeping an open mind to the possibility that the project you're invested in may be the lower priority.
- **Frame the problem**. Ensure that all parties recognize the scope of work that is being requested. If the scope of work is small enough, perhaps a team can be persuaded to deliver work it considers to be lower priority to unblock the other teams, before embarking on their higher priority work.
- **Recruit an away team** - Can engineers be lent to the partner team's higher priority work, freeing up resources for them to work on your priority?
- **Escalate**. Ask leadership to clarify which is a higher priority for the business.

**Scenario**

As the discussion around the architectural approach for the new home renovation experience heats up, the conflict between the two opposing viewpoints — sticking with the existing monolith versus breaking the functionality into microservices — becomes more pronounced. The backend teams are firm in their belief that the monolithic architecture, though older, provides a stable foundation and allows the project to meet immediate deadlines. Your teams on the other hand argue that breaking the features into microservices will ensure future flexibility and better support the expected growth of the platform.

Understanding that this conflict could stall the project and create rifts between the teams, you decide to step in and mediate the discussion. Your first step is to *understand the stakeholder motivations* behind each stance. You schedule one-on-one discussions with the key representatives from both sides to get a clearer picture of their concerns. The backend teams are primarily concerned with delivery timelines and minimizing disruption to the current user experience. Your teams are more concerned with the long-term technical debt and future-proofing the system. This insight helps you see that the disagreement is not just about the

architecture itself but also about differing priorities — short-term delivery versus long-term maintainability.

With this understanding, you bring the group back together and *frame the problem* in a way that highlights the underlying concerns. You articulate the challenge not only as a choice between two architectures but as a broader issue of balancing immediate business needs with future scalability and maintainability. This helps to depersonalize the debate and focus on the core issues at hand. You help the teams see that the decision is not only a technical one but also a strategic business decision that requires weighing various trade-offs.

Next, you propose to *decompose the problem*: instead of deciding the entire architecture upfront, the working group could focus on breaking down the most critical components for the new home renovation experience, such as user authentication and appointment scheduling, into microservices while keeping other, less critical functionality within the monolith. This hybrid approach serves as a compromise, allowing teams to begin with a more scalable architecture in critical areas while maintaining stability and speed of delivery. By breaking down the problem you've reduced the perceived upfront cost and made the migration feel more manageable.

As the meeting concludes, the tension eases and you guide the group toward a consensus of moving forward with a hybrid approach. In resolving this conflict you've kept the project on track and maintained positive working relationships between the teams by honoring their individual perspectives. You've leveraged your influence to resolve a technical dispute while balancing both the technical and business needs of the group — and modeled effective collaborative problem-solving.

## **Technique:** Communicate with clarity

A principal engineer's ability to influence depends on how clearly they communicate. Whether you're writing a document or speaking in person, strong communication skills are key to conveying complex technical concepts to diverse audiences, driving alignment across disparate groups, and facilitating collaborative decision-making.

**Four techniques for effective communication**

1. **Tailor your message to your audience**. Before you craft your communication, have a target audience in mind and understand their unique perspective and needs. Frame your message in a way that will resonate with them. For example, when communicating with:
    - **Senior leadership**, focus on the big picture, emphasizing the impact and return on investment. Address risk and mitigation strategies. Quantify the costs, including team capacity. Leadership's time is especially valuable, so be succinct, to the point, and leave room for questions.
    - **Engineers**, explain the why and how, providing detailed information that encourages feedback and collaborative problem-solving. Engage engineers in crafting the communication when possible to ensure thorough understanding and buy-in.
    - **Product managers**, speak to the user impact and business value. Simplify technical jargon and concepts, highlighting how the technical aspects support the product vision and address user needs. When discussing trade-offs, address not only user outcomes but also the impact on delivery schedules and timelines.
2. **Use the inverted pyramid**. This concept originates in journalism, and refers to starting with the most important idea (typically the purpose or objective of your communication), followed by the next-most crucial concepts, such as background information and context. Continue to present ideas in order of diminishing importance, ending with the most specific details. This approach ensures that your audience grasps the core message quickly. It's especially useful in written communication since readers may stop after reading the first few lines if it's unclear how the content is relevant to them.
3. **Leverage data to persuade**. Use data to weave a compelling narrative that informs and persuades your audience. Present clear, concise metrics to support your arguments, explaining how they were collected and the context to interpret them accurately. Visual aids like charts and graphs can be particularly effective in illustrating key points and trends.
4. **Encourage feedback**. Create opportunities for your audience to ask questions and provide feedback. This not only enhances their

understanding but also provides you with valuable perspectives that can better inform the outcome you hope to achieve.

**Scenario**

With the general approach defined and some preliminary technical design outlined, it's time to update engineering leadership on the direction and progress of the working group. You set out to craft a presentation that effectively communicates the working group's decisions and future plans. This presentation will be crucial in ensuring that leadership understands the strategic direction, supports the next steps, and provides any necessary feedback to guide the project forward.

You begin by carefully *tailoring your message to your audience*. Engineering leadership is primarily concerned with the big picture: how the project aligns with overall business objectives, the risks involved, and whether the team is on track to deliver value. With this in mind, you focus on presenting the strategic decisions made by the working group, such as the hybrid approach to architecture and how this will enable timely delivery and long-term maintainability.

You structure your presentation using the *inverted pyramid* technique. You start with the most critical information: the working group has aligned on a hybrid architectural approach. You explain that the team has made initial progress on designing core features, and you highlight how these decisions align with the business objectives. Next you provide the context that led to these decisions, the technical conflict that was resolved, the trade-offs considered, and the rationale behind the chosen architecture. By framing the problem and solution clearly, you help leadership understand the thought process behind the decisions and build confidence in the direction the project is taking. Finally, you dive into the specific details, such as major design and development milestones and the metrics the project will use to evaluate success. You ensure that the more detailed design outlines are available for those who want to dig deeper, but keep the focus on the strategic overview, knowing that this is where leadership's primary interest lies.

Throughout your presentation, you actively *encourage feedback* from the leadership team. At the end of the presentation, you invite them to share any remaining concerns or questions they may have, demonstrating a commitment to transparency and creating an opportunity to gather valuable insights.

As the presentation concludes, the leadership team expresses their appreciation for the clear and concise update. They feel confident in the direction the project is taking and value your willingness to seek their input. By communicating clearly and effectively, you've not only provided leadership with a successful update but also strengthened your own influence with the leadership team.

## **Technique:** Collaborative writing for shared understanding

Principal engineers often lead complex projects that need input from many different voices, each with their own expertise and perspective. Asynchronous collaborative writing is an essential tool for bringing those voices together and creating shared understanding. By shaping how ideas are captured, shared, and refined, you can play a pivotal role in aligning teams around common goals and driving consensus.

Collaborative documents — such as design docs, decision memos, or project briefs — are powerful tools for aligning teams, clarifying intent, and capturing institutional knowledge. Your ability to guide these writing efforts is critical to influencing others and leading high-impact work.

Today, large language models (LLMs) can help streamline this process. Use them to draft initial outlines, reframe complex ideas, or rephrase content for different audiences. When used thoughtfully, these tools can make your writing more clear, structured, and inclusive — especially in early drafts, when shared understanding matters most.

Following are some other tips which we have found helpful when authoring documents for a shared audience.

- **Have a single pen holder**. Assign a directly responsible individual (DRI) to oversee the document. This person ensures consistency in structure and voice, resolves disputes, and keeps the document development on track.
- **Have clear areas of ownership**. When different sections of the document require different ownership (e.g. a broad design document that spans several organizations), clearly indicate ownership of each section. This helps prevent overlap, ensures accountability, and allows contributors to focus on their specific area of expertise. Linking out to more detailed documents can also provide depth without cluttering the main document.

- **Set deadlines**. Establish clear deadlines for each phase of the document's development: initial drafts, feedback, revisions, and finalization. Document these commitments and communicate them clearly to stakeholders.
- **Indicate document status**. At the top of the document indicate whether the document is a "work in progress" or "requesting feedback." This transparency helps manage expectations and guides readers on how they should engage with the content.
- **Be responsive**. Address questions and feedback promptly. Resolve comments to keep the document from getting too cluttered, but only after the feedback has been incorporated into the main document so it can first benefit those with similar queries.
- **Follow up**. If you're not getting the engagement you expect, reach out to relevant stakeholders. Sometimes scheduling a meeting to discuss and resolve lingering issues can be more effective than waiting for written responses.
- **Subscribe to notifications**. Sign up for notifications of document or comment changes and encourage contributors to do the same. In Google Docs, for example, you can  make this change in your notification preferences.
- **Leverage permissions effectively**. Start with broad access for feedback, then tighten permissions as the document nears completion. Alternatively, you might begin with a small group of reviewers, opening it up more broadly after the general structure has firmed up. Adjusting permissions strategically ensures that the right people are involved at the right time.

| Exercise | Think about a time when you tried to influence decision making in a group that spanned many areas of ownership. What techniques did you use? Were you successful? If not, why do you think that is? |
|---|---|

Part IV

# Scaling yourself

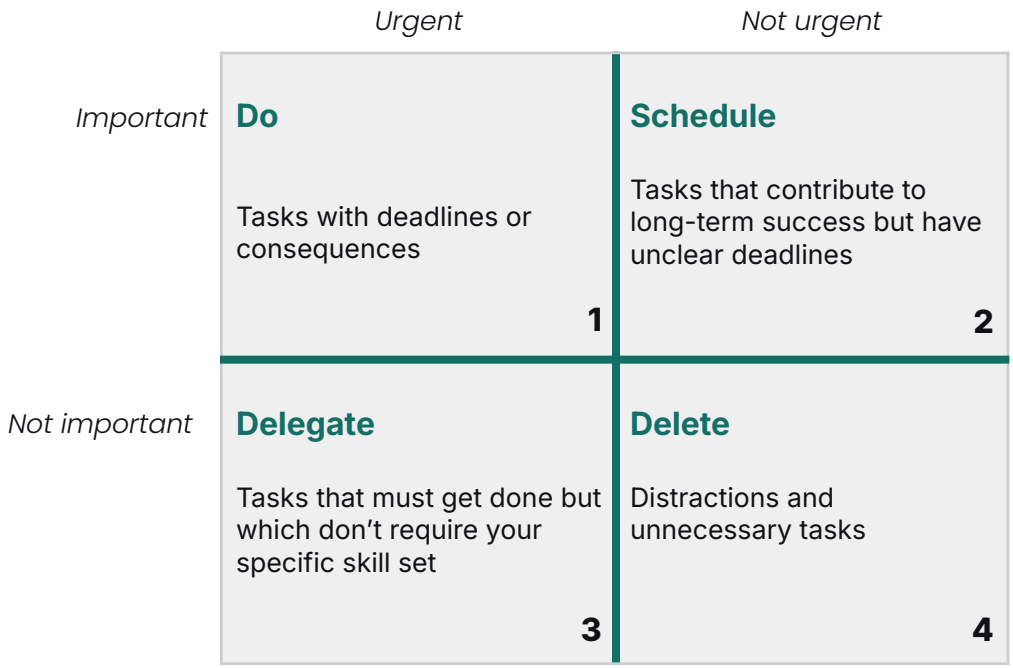
> *"Impact is paramount, and you must scale. Sometimes, your impact will be greatest when you go deep to author critical code or uncover a subtle flaw; at other times, your impact will be greatest when you act broadly to shape approaches across several teams or systems."*
>
> *"You understand that your impact scales with your influence: you cannot uphold these standards solo, but instead must rely on the team to embody them even when you're not looking."* Leadership facet

— *Zillow Leveling Guide*

*Scaling yourself* as a principal means maximizing your impact without overstretching your scarce time, energy, and influence. Operating at larger scales can be challenging: ambiguity increases, trade-offs become more difficult to identify, and feedback loops lengthen.

The Eisenhower matrix, also known as the urgent-important matrix, is a powerful tool for prioritizing tasks and managing time effectively. Named after Dwight D. Eisenhower, the 34th President of the United States, this matrix helps you focus on what matters by categorizing tasks based on:

- **Urgency:** When the task needs to be done.
- **Importance:** The value delivered by accomplishing the task.




The matrix is divided into four quadrants:

1. *Urgent* and *Important* – **Do**
   Tasks in this quadrant require immediate attention and are critical to achieving significant organizational goals. These are often crises, deadlines, or problems that need to be resolved promptly, such as ongoing operational events or end-of-quarter deliverables.

2. *Important* but *Not urgent* – **Schedule**
   These tasks are crucial for long-term success but do not require immediate action. Activities here include many of those discussed throughout this book: strategic planning, relationship building, long-term thinking, building communities of technical excellence, and so on.

3. *Urgent* but *Not important* – **Delegate**
   Tasks in this quadrant are often distractions that require immediate attention but do not contribute significantly to long-term goals. These can include interruptions, most emails, Slack reach-outs, and meetings that do not add

substantial value. These are the tasks that promote context switching — which is the death of efficiency, as we have to rebuild context when we return to the task at hand. We'll speak to techniques to manage these kinds of tasks, including delegation, focus time, and saying no.

4. Neither *Urgent* nor *Important* – **Delete**
   These tasks are time-wasters and should be minimized or eliminated. Sometimes, these are easy to identify (checking social media), but usually they are tasks that are misclassified into quadrant three: tasks that we consider more important than they actually are. Careful scrutiny of tasks in quadrant three can save you time — move them into quadrant four and drop them.

A principal's ability to achieve long term impact will be a function of their ability to prioritize quadrant two over quadrants three and four, while curating quadrant one. Quadrant two is where we spend time thinking strategically, where we can take the time to evaluate, consider, and innovate. Because this work is rarely urgent and the value more speculative (especially in the short term), it's easy to procrastinate quadrant two tasks in favor of less-important but higher-urgency issues. You might choose to help a junior with a minor MR or read a paper where you were added as the third or fourth reviewer — tasks where you can quickly add value — making it easy to justify putting off longer-term strategic work to tomorrow.

One of your biggest advantages as a principal engineer is that, unlike managers, you aren't constantly pulled into the daily demands of running an organization. That gives you space to focus on the strategic, long-term work in quadrant two — if you don't make the time for it, chances are no one in the organization will.

We'll walk through the Eisenhower matrix quadrant by quadrant, considering different techniques and practical methods to help you protect your time, prioritize where you spend it, and get the most out of it.

| Exercise | Take a moment to write down your current Eisenhower matrix. What tasks are on your plate right now? What tasks have you wanted to work on but struggled to prioritize? |
| --- | --- |

### Quadrant 1

# Getting the important things done

| |
|---|
| *"You deliver critical business value by solving complex problems that span teams and organizations. You partner with fellow leaders to identify problems and align on their nature."* Principal engineer summary |
| *"You are trusted to exercise your judgment—you take the initiative, and you are skilled at communicating why, what, and how to your leadership."* Principal engineer summary |
| *"You cultivate collaboration, listening to and integrating feedback across all levels."* Leadership facet |

*— Zillow Leveling Guide*

The work that falls into quadrant one is the critical stuff. Work in this box will crowd out other work, especially the long-term work in quadrant two. First, you must keep this box as lean as possible, ruthlessly prioritizing only those tasks that are both truly important (will create true impact rather than be perceived as important) as well as actually urgent (the value is truly time-sensitive). Second, you must be efficient when you set about actually doing the work.

## **Technique:** Discuss priority with stakeholders

Your many stakeholders will attach different priorities — different senses of urgency and importance — to different tasks. The more senior your role and the greater your influence, the more stakeholders you'll have to manage. You can't please everyone, and you shouldn't try to. Rather, you should work closely with your manager to develop your own priorities and share these priorities with your stakeholders.

One method for doing this is periodically discussing your Eisenhower matrix with your manager. Share it with your peers and stakeholders, too, for feedback. Their perspectives will help inform (and check) your classifications. Stakeholders can be surprisingly reasonable when they see their tasks compared and contrasted with others on your plate.

In particular, you should seek their advice on which tasks in quadrant one can be moved right to quadrant two or down to quadrant three.

**To determine whether a task can be shifted right**, ask questions such as:

- What happens if this work is delayed a week? A month? A quarter?
- Will irreversible (or hard to reverse) decisions be made if you don't act ASAP?
- Can the work be rearranged or re-thought to disentangle the urgent from the non-urgent?
- What's the opportunity cost of delaying this work compared to other work in the matrix?

If a task doesn't turn out to be as urgent as originally communicated (or as urgent as you originally assumed), move it to quadrant two.

**To determine whether a task can be shifted down**, ask questions such as:

- What's the risk of letting someone else do the work?
- Is the task expensive to do myself but easy for me to audit someone else's work?
- Does this work present a growth opportunity for a junior? Is there a junior well equipped and eager to do this work?

If a task can be done *well enough* by someone else, move it to quadrant three. By considering these questions hand-in-hand with your manager (or with peers in the organization such as SDMs or other key engineers), you may be able to quickly agree that certain tasks should be moved off your plate. For example, you may move a task from quadrant one to quadrant three, doing the work necessary for an effective hand-off. We'll discuss delegation in more detail below.

### Antipattern

Few things feel as rewarding as being the superhero who dives in to "save the day" — no one else could do the work correctly and on time, only *you!* But this impact is one-off and fleeting. To create enduring, scalable impact, you must remove the need for heroism in the first place. It's better to be the fire marshal preventing fires through inspections than the fire fighter charging into the fire after the fact — even if the latter gets the glory. Being the marshal requires us to reflect on quadrant one: why are these essential and why can't they be delegated or handled later?

Remember not to conflate visibility with importance. Seek to delegate low-impact but highly-visible work to a more junior engineer. As a principal, you naturally have more opportunities for visibility, and delegating these tasks allows others to gain exposure while freeing you to focus on other work.

**To determine whether a task can move from quadrant one to four**, there's only one question to ask:

- What happens if we simply don't do this?

It's common to overvalue a task, especially when the output is for someone else. By talking to the supposed beneficiary up front, we might find that they are more than willing to drop the request in favor of your other priorities. For example, you may have agreed several months prior to attend a weekly ops review, but the value of your frequent participation may have dwindled over time, and you might decide to attend less frequently.

Keep in mind that it's easy for others to assume you have additional capacity when you simply don't. Review this matrix with your manager and stakeholders, frequently.

| Exercise | Bring your Eisenhower Matrix to your next 1:1 with your manager. Talk through it. How did this conversation go? Were you able to have a more productive conversation around how to prioritize your time? If not, why not? |
|---|---|

## **Technique:** Planned focus time

Once we've whittled down to the truly important tasks, we still have to *actually do the work*. As a principal, you must dive deep, think carefully, weigh details, consider long-term implications, and creatively solve problems — all of which require concentration. Your time is limited, and full of competing demands. If you only try to squeeze strategic work in between other commitments, you'll struggle to do it well. Instead, proactively plan for dedicated focus time.

Granted, this is easier said than done. Here are a few strategies to help protect that time:

- **Create calendar blocks.** Put this time on your calendar just like any other meeting. This helps signal to others that you are unavailable during those times. It's on you to try to treat these as non-negotiable commitments to yourself.
- **Establish a recurring rhythm.** Calendar blocks can be more effective when they follow a consistent pattern. For example, you might reserve the first two hours of each morning, or designate one or two days a week for deep focus. A predictable rhythm makes it easier for others to learn when not to interrupt you, reducing the likelihood of ad hoc meeting requests during those windows. It also helps you build a habit. When focus time becomes routine, it's easier to get into a deep work mindset quickly.
- **Set expectations on Slack**. Set your status to 'Slow to respond' or 'Focus time' to set expectations with others. Then, mute Slack notifications. This will help you focus and  put your mind at ease by removing the obligation to respond quickly, and removing distractions.

**Actually focus:** Once you've claimed focus time, you better make the most of it! Do whatever you need to do to focus: minimize distractions and set clear goals.

For some individuals, this is sufficient: given uninterrupted time, they'll dive right in and get the work done. Not all of us find this easy to do, however. There are many frameworks and methodologies to help you make the most of your focus time. Whether one will work for you is a matter of personal preference.

**Here are two methods you can explore if you need more help structuring your focus time:**

- **Flowtime technique** — A time management approach designed to accommodate tasks that are harder to estimate and often depend on creativity (such as writing, coding, etc).
- **Time blocking / Time boxing** — A method of creating deadlines for yourself by visually blocking out time (on a calendar or equivalent).

## **Technique:** Manage your meetings

We often default to treating meetings as ‘must-do’ tasks, letting them fill up quadrant one. Protect your time by avoiding unnecessary meetings and making the necessary ones efficient and effective. First let’s look at methods for getting the most out of meetings you control.

When meetings are essential, it’s your responsibility to ensure they are productive and meaningful. Here are key strategies for maximizing meeting effectiveness:

- **Invite only the necessary people**. Large meetings tend to waste time and create unnecessary noise. Limit attendees to only those who need to be there to either contribute to or act on the outcomes.
- **Start with a clear agenda**. Ensure that every meeting has a defined purpose and agenda shared in advance. When participants know the objectives, they come prepared, leading to more efficient discussions. A clear agenda helps everyone stay on track and eliminates unnecessary conversations.
- **Set time limits and stick to them**. Time-box your meetings based on the agenda. Set the meeting duration based on the actual discussion time needed. Start on time and end promptly, respecting everyone’s schedule.
- **Facilitate focused discussions**. Keep discussions on point, and don't be afraid to steer the conversation back to the agenda if it derails. That said, remain open to improvisation when valuable ideas arise—balance preparation with flexibility.
- **Summarize and assign action items**. At the end of the meeting, summarize key takeaways and ensure clear action items with owners and deadlines. This helps prevent the need for follow-up meetings and holds everyone accountable for moving forward.

In general, the smaller the invitee list on an invite, the more important a role you’ll play in driving value at the meeting. But paradoxically, it’s easy to fall prey to the fear of missing out when the invitee list is large: if you don’t go, what will you miss? Will key information be shared that you won’t be privy to? Will some statement be made that only you can respond to or correct?

Don’t fall into the trap of attending meetings out of FOMO. Insist that organizers provide post-meeting summaries (whether manual or AI-generated), and that documents be shared in advance with potential decisions outlined in the agenda so you have a chance to assess the value of your attendance beforehand. Don’t go just for the sake of including your face on the Zoom mosaic.

**Antipattern**

One of the most common mistakes in meeting culture is scheduling a meeting just to discuss another meeting. This happens when you haven't prepared enough or lack clarity on the objectives. Instead of having meaningful discussions, time is wasted rehashing the agenda, defining roles, or planning yet another follow-up. This "meeting for the meeting" creates unnecessary friction, delays decisions, and ultimately causes meeting fatigue.

What about meetings you don’t control? Carefully evaluate requests for your time and learn to avoid saying yes to unnecessary meetings. Avoiding unnecessary meetings isn’t just about saving time by canceling — it’s about choosing the most effective way to communicate and understand your audience. Be flexible in your approach, meet people where they are, and choose the method that best fits the situation and the habits of your team. Reserve meetings for when high-bandwidth verbal communication is most valuable, such as working through disagreements in approach, discussing a nuanced issue, or building a social connection.

Consider these alternatives to a meeting:

- **Impromptu huddle**: Call for a brief, no-video chat when quick alignment is needed. These huddles keep interactions focused and get right to the point, saving time.
- **Slack conversation**: Use Slack for quick back-and-forth or to gather input asynchronously without disrupting schedules, keeping conversations efficient and organized.
- **Written document**: Collaborate on a shared document to gather feedback or input, providing a clear, asynchronous way to engage with complex topics.
- **Email**: Use email for updates, requests for feedback, or progress reports that don’t require immediate responses, ensuring clarity without interrupting workflow.

### Quadrant 2

# Working on the long-term things

> *"The feedback cycles are longer than before, stretching across quarters and into years."* Strategy facet
>
> *"You are expected to identify, define, socialize, and break down novel problems in your organizational domain."* Strategy facet

*— Zillow Leveling Guide*

Principal engineers must find time to consider the long-term. We are best positioned to explore a new architectural direction, realign work across teams and organizations to better account for Conway's Law, drive engineering excellence initiatives, or investigate a new technology. Yet, these activities are easy for us to deprioritize in the face of more urgent tasks.

How do we find the time to work on these and how do we select the right ones to work on?

The kinds of tasks that tend to fall into quadrant two aren't urgent and, because they often require much thought and discussion, you can take your time to figure out which ones are really worth investing in.

Towards that end, we suggest thinking about the tasks in this quadrant as "irons in the fire." Many ideas *might* be worth investing further in: Use the following techniques to filter through and find the ones that you are ready to act upon.

## **Technique:** Let time reveal what's important

Time offers perspective. What appears to be absolutely necessary today often turns out to be nice-to-have or, even, completely off the mark within months. Our own career experiences as principal engineers help us peer past this veil, but it's worth emphasizing  that, often, the best course of action is to watch and wait.

Delaying an action doesn't mean doing nothing. We use this time to collect information and let ideas evolve. Keep a private running document to gather ideas, links, notes, and insights as they come. Let the idea take shape gradually. Leverage casual and low-pressure conversations outside of structured meetings and looming deadlines to test the ideas with others.

By taking time to explore an idea patiently through casual conversations, you not only create the opportunity to convince *yourself* that an idea is worth pursuing, you have the opportunity to convince others as well. By the time you're ready to write a doc or pitch the idea more formally, your idea will feel familiar to your audience. You'll meet agreement, not resistance.

## **Technique:** Write it down

A more time-intensive approach to exploring and collecting information on an idea is to write it down as a request for comment, or RFC. The act of writing will help you formalize your thoughts and think through the problem. A well-written doc walks others through your reasoning and thought process. If people respond — asking questions, sharing feedback, or building on it — that's a signal that this is worth pursuing further.

This approach requires you to adopt a mindset of genuine curiosity: seeking feedback rather than pushing for change. Let the document drive conversations. Let it develop over time. Let others become co-authors or contributors. Let the document become a vehicle for sharing ownership — after all, when others feel shared ownership of an idea, they're more likely to champion it..

**Additional benefits to writing down your thoughts**

1. **The document keeps working when you're away from it.** You can be stuck in a meeting, or busy driving a quadrant one task to completion, while others are reading your document and recognizing the patterns you've described. Someone who finds your document compelling can share it with someone else without any distortion — unlike verbal conversations which can lose clarity as they pass from one person to the next.
2. **The document is a place to recontextualize and re-energize yourself.** For many of us, this kind of long-term problem solving — ruminating on difficult,

ambiguous problems — is our favorite kind of work. It energizes us,and re-energizes us when we get to return to it after our day-to-day routines, meetings, and immediate deliverables. A living document helps you pick up this work as time allows, quickly rebuilding context instead of restarting from scratch. Note that 'document' here need not be a literal Google document — it could be a whiteboard in your office, a sketch in Excalidraw, or something equivalent that works well for *you*.

### **Technique:** A bi-modal approach to decision making

There are often two points in time when it's ideal to make a decision: as early as possible or as late as possible. Acting early promotes agility and speed — qualities that can fade as organizations grow more complex or risk-averse — and primes the business to nimbly respond to changing information and market conditions. On the other hand, delaying a decision allows for more time to gather information, which leads to more informed decision-making.

Delaying a decision doesn't imply passivity. Rather, it encourages using techniques like pilots, MVPs, and incremental experimentation to actively gather information. Each of these investigative tasks can be treated as quadrant one tasks, prioritized ahead of the final decisioning task in quadrant two. Said another way, one can decompose quadrant two work into a series of more urgent tasks that will reduce uncertainty without delaying progress inordinately.

| Exercise | Is there a quadrant two task you've been meaning to tackle, but haven't made real progress on yet? How might you begin to gather more information, collect your thoughts, and share this idea with others? |
|---|---|

## Quadrant 3

# Shedding the unimportant things

| |
|---|
| *"Impact is paramount, and you must scale."* Strategy facet |
| *"When you identify workstreams in large projects, you guide juniors to effective solutions and inspect their estimates."* Leadership facet |

*— Zillow Leveling Guide*

Tasks in quadrant three will fall into two buckets: the less-than-important tasks that you must do for reasons beyond your control, and those less-than-important tasks that you've brought upon yourself.

You have little control over the former, short of what your management chain can do to pull things off your plate. We all have the sorts of tasks that matter to the business and, in aggregate, are important — bookkeeping, large meetings, and training — but limited immediate value to us. You just have to get through them.

The latter kind is more interesting — and more solvable. These tasks often come from situations that could have been prevented: unclear ownership, architectural debt, overloaded systems, or overcommitments. Perhaps the juniors in your organization were ill-equipped to pick up slack, or perhaps you simply tried to fix too many things and took on too much work. Diagnosing these issues requires context, judgment, and self-reflection — which fall to you as the principal. We have three techniques to help you succeed.

## **Technique:** Delegation

Tasks in quadrant three might be an opportunity to grow others in your organization. The deeper the technical depth and leadership in your organization, the more often you can hand off a task in quadrant three to someone else — freeing you up to focus on quadrants one and two. And the more often you delegate a task in this quadrant, the more effectively you build organizational depth and capability.

Successful delegation is not the process of reassigning work from yourself to others. While you can go to your manager and ask for tasks to be handed off — and sometimes this might be necessary — it's not ideal. You're likely to suffer pushback from others (juniors, the SDMs they report to, etc.) and resentment is likely to build. The engineer who gets the work handed to them is probably

unsympathetic to a story like, “Well, I have a lot to do” — so do they! The art of delegation requires some prep work to be successful.

If you’ve built great relationships with developers across the org, you should have a good sense of who will benefit from what kinds of work. Who is seeking a stretch project that aligns well with this task? Who simply enjoys this kind of work? Who has ideas around automating away this kind of task? And so on.

Leverage your relationships to float the idea of a hand-off to the individual. Get them interested, even excited. The most successful acts of delegation are never perceived to be delegation at all: the individual *asks* to do the work rather than being *told*.

At minimum, coming to management (whether peer SDMs or your manager) with a suggestion rather than a request will improve your odds of delegating. If you can suggest who is better positioned to pick up the work — and why — you’re much more likely to get the work off your plate.

The follow-through matters as well, of course. Refer back to part one to refresh yourself on how to guide, how to inspect, and how and when to let others fail.

**Scenario**
As a principal engineer working closely with the Data Platform team, you're leading a major infrastructure upgrade to improve data pipelines. However, your calendar is packed with high-priority tasks, and it’s becoming clear that you need to delegate some work. One task involves refining a set of monitoring scripts for the data pipelines — important, but something that doesn’t demand your specific skills.

You decide to leverage your relationships across the organization. You know that Alex, a senior engineer on one of the data teams, has been looking for more opportunities to take on stretch assignments. They’ve shown an interest in monitoring systems and have even mentioned wanting to dive deeper into automation work. Recognizing this, you decide this would be a great task to delegate to them.

You approach Alex and have a conversation about the project. You ask them for their thoughts on improving the current monitoring setup and whether they see any opportunities for automation. As the conversation develops, Alex begins

sharing some ideas they've had on streamlining the monitoring process. You guide their thinking with questions, helping them see the larger implications of the task and how their work could impact the entire pipeline.

By the end of the discussion, Alex is excited about the challenge and asks if they can take ownership of the task. You agree, handing over this work without making it feel like an assignment. You then take the additional step of mentioning this to their manager, making them aware that this is an opportunity for Alex's growth and explaining how their strengths align with the task. The manager appreciates the thoughtfulness and supports Alex's involvement.

Over the next few days, you check in periodically to inspect Alex's progress, offering feedback and guidance when necessary. When Alex hits a roadblock, you encourage them to think through the problem and propose solutions before jumping in to help. This gives Alex the space to learn and grow, while ensuring the project stays on track.

The result is twofold: Alex successfully completes the task, improving the monitoring system while gaining new skills and confidence. Meanwhile, you've freed yourself to focus on more critical strategic work, all while strengthening relationships and building depth within the org.

### **Technique:** Reduce planning commitment

Part of delegation is stepping back from routine ceremonies and planning work that senior engineers are capable of doing on their own. If a team is healthy, a principal engineer's presence shouldn't be needed for part planning, task estimation, stand-ups, and backlog grooming. Unless your involvement will clearly unlock progress — because of unique expertise or a high-risk situation — you shouldn't sit on the critical path for defining and divvying up small units of work.

Show up when it matters. Maybe the team is junior and needs guidance. Maybe a project is off track. Maybe you're checking in to nurture relationships and stay connected to the organization. These are all valid reasons. Just be intentional with your time and don't let ceremony control your calendar.

## **Technique:** Identify systemic causes

Tasks in quadrant three might be indicative of systemic issues. Ops tasks piling up? Might be a sign of systemic architectural or cultural problems. Small product changes require last-minute scrambling (design, configuration, etc.)? Might be a sign of architectural limitations or a failure to align with stakeholders. Requests to explain tough problems — infrastructure costs, RCA trends, ops dashboards, etc. — piling up and randomizing you? Might be a sign of a lack of engineering expertise across your organization. And so on.

Identifying and diagnosing these patterns may not provide immediate value. You're identifying new tasks for quadrant two that can generate lasting long-term value. This is exactly the sort of work that principal engineers are equipped to do — identify systemic issues from trends and articulate the problem to leadership. Use the Eisenhower matrix to drive the discussion with your manager when you're trying to prioritize quadrant two work: investments there will drive down time spent in quadrant three over the long term.

## **Technique:** Choose your battles

Principal engineers have the experience to identify problems better than the juniors around them. On one hand, this is good, and it's key to the value you bring to the organization. Yet, from your elevated vantage point you are likely to see more problems than ever before. So many problems. Too many problems.

As you progress in your career, it becomes increasingly important to recognize that you can't do everything. The key to managing your workload effectively is to focus on the most high-leverage areas. This means ignoring legitimate problems — or, at least, treating these problems with a lesser sense of urgency than your first instinct might suggest.

Ideally, we gate this kind of work before it ever ends up on our Eisenhower Matrix: we saw the problem, decided the problem wasn't worth solving (or the time wasn't ripe), and never took on the work of solving it. Yet, in practice, some will inevitably slip through. It's healthy, therefore, to re-ask the question, "What happens if we simply don't do this?"

If the answer is “this problem can wait,” avoid simply shifting it to quadrant two. You already decided it was *unimportant*, so don’t categorize its importance just to keep it on the matrix. Move it to quadrant four.

| Exercise | Review your quadrant three tasks. Which ones might you be able to delegate? Of the remainder, how might you reduce the number/frequency of these in the future? |
| --- | --- |

### Quadrant 4

## Discarding the rest

> *“You demonstrate effective verbal and written skills, communicating with everyone from the junior-most engineer to the senior leaders of your organization.”* **Communication facet**
>
> *— Zillow Leveling Guide*

Too often, the Eisenhower matrix is presented in a way that makes quadrant four appear to be the easy bit: if a task lands here, delete it! Move on with your life! If it doesn’t bring you joy, so to speak, toss it in the bin!

In practice, however, a principal engineer rarely has this luxury. Each task on our plate has stakeholders, and those stakeholders want the work done for *some* reason. Even if they agree it's *less* important than the other stuff on your plate, they won’t be keen to see it dropped, either.

In truth, the best way to handle these kinds of tasks is to keep them off your Eisenhower matrix in the first place. When a task lands in this quadrant, reflect on how it got there, and strategize with your manager on how to minimize such tasks in the future.

We'll offer up a couple techniques to keep these off your plate in the first place. You must learn how to say *no* to requests. This is easier said than done. Different stakeholders may require different ways to be told *no*. Here are a few suggestions.

## **Technique:** Saying no to your managerial chain

A simple *no* to a request from your manager or skip isn't going to cut it — they'll raise an eyebrow and repeat the request. Yet, our managers don't expect us to be heroes. They expect us to discuss priorities. We have a finite amount of capacity, so a new request requires something else to be deprioritized.

Don't make the mistake of asking your manager an open-ended question like, "So what should I drop?" Exercise your judgment instead (perhaps leveraging a model like the Eisenhower matrix) to proactively suggest which work should be deprioritized. This helps your manager make a quick decision around whether the new work is worth the trade-off, and it helps you better influence these kinds of top-down requests.

**Scenario**
You're the principal engineer overseeing Zillow.com's ongoing effort to improve page load performance — a critical initiative aimed at enhancing user experience and SEO rankings. As your team is deep in this project, your manager approaches you with a new request: the marketing team wants to add a complex, interactive map feature to the homepage to highlight trending neighborhoods, believing it will boost user engagement.

While the map feature aligns with business goals, the amount of design and development required would pull key resources from the performance improvements, threatening your team's current focus. Instead of simply saying "no," you use the Eisenhower matrix to frame the conversation.

In your next 1:1, you acknowledge the potential impact of the map feature but explain that it falls into the "important but not urgent" category compared to the performance initiative, which is both urgent and important. You note that improving load performance has a more immediate and broad impact on users across the entire site, whereas the interactive map feature *could* be impactful but not at the same scale or urgency.

Rather than just presenting the problem, you suggest a solution based on priority: the map feature could be delivered in phases. In the first phase, a simplified version showing trending homes as a clickable list — using existing APIs and designs. This allows your team to maintain focus on the performance work with minimal disruption. Once the performance project is complete, your team can revisit the trending homes feature and implement the more complex map and interactive elements in future iterations.

Your manager appreciates the structured approach and your ability to prioritize effectively. By showing how the map feature fits into a broader strategic plan and balancing it with urgent tasks, you address the request without compromising the performance work. The phased solution allows the marketing team to get an early version of the home trends feature while your team continues with the high-priority performance improvements.

## **Technique:** Saying no to a stakeholder

Unlike your manager, a stakeholder (PM, TPM, engineer from another organization, etc.) isn't necessarily interested in the holistic view of where you spend your time — given the option to deprioritize someone else's task in favor of their own, they'll tend to take it every time.

The trick here is to avoid saying yes when the request is first made. Promise to get back to the requester with a decision in a timely fashion. Then, take the time to evaluate this request against your other tasks. Engage your manager as necessary. If the answer is no, politely decline (perhaps cc'ing your manager over email or including your manager in a Slack message, if necessary). Or, work with your manager to delegate the requested task to someone else.

If the request is for your participation in a meeting, ask to see a clear agenda up front. Will you add value or is the meeting creator simply trying to build a complete list of possible stakeholders? Request documents up front for review and, if you have nothing to add in a synchronous context, politely decline the meeting. Encourage asynchronous alternatives where possible to ensure meetings stay productive and relevant.

## **Technique:** Saying no to a junior

As we grow more senior — and as we build better relationships with juniors — requests for our mentorship and guidance will grow. These requests are very easy to say no to, in the sense that a junior has little recourse to insist that you say yes. Yet, it is for this very reason that we should avoid taking the path of least resistance by turning these requests down just to preserve space for managerial or stakeholder requests.

When we must say no to juniors, we don't want to inadvertently discourage them from making such requests in the future, whether to us or other seniors. Thank the junior for reaching out, clarify why they reached out, and what they hoped to get out of the request, and offer them asynchronous guidance as best you can. Methods that might be applicable, depending on the situation:

- **Suggest someone else**. This could be another principal engineer, a relevant manager, or another junior engineer who is well-equipped to handle the request at hand. Ideally, you'll pave the path by reaching out on their behalf; perhaps in a shared email chain or Slack thread
- **Suggest resources.** As principals, we often have a bookshelf (real or figurative) of books, articles, and talks that we've found helpful in our career. Point the junior to relevant resources that they can engage with independently.
- **Promise a backstop.** You can increase psychological safety by promising to make the time in the future if other methods don't pan out. They can investigate a problem on their own, knowing that if they don't make progress (or if other individuals prove less than useful), you'll be there to step in and help them out. Many such requests will simply go away after a week or two.

| Exercise | Reflect on your quadrant four tasks. How might you go about declining such tasks in the future? Reflect on your calendar. Which meetings might be classified as quadrant four tasks? Which ones should you politely decline? |
|---|---|

Part V

# Building technical strategy

Technical strategy outlines how technology will be used to meet long-term business goals. It involves understanding business priorities, reducing ambiguity, leveraging proven technologies, and balancing short-term deliverables with long-term vision to ensure that the technical strategy is both resilient and adaptable.

Next, we'll discuss the fundamentals of building a good technical strategy. We'll explore how to align our technical efforts with business priorities, balance short-term and long-term goals, and leverage industry best practices. By the end of this part, you'll be better able to develop and articulate a technical strategy that not only addresses immediate needs but also paves the way for sustainable, long-term success.

We'll also discuss techniques such as steel threads, strangler fig re-architecture, A/B testing, and MVPs to ensure that our strategy is not just a theoretical construct but a practical, actionable plan. You'll learn about transforming high-level strategic concepts into concrete actions that drive business value. By understanding and applying the principles of good strategy, we can navigate the complexities of our technical landscape and achieve our long-term vision.

Section 1

## Fundamentals of good strategy

In software development, a robust technical strategy helps teams navigate the complexities of modern business challenges. According to Richard Rumelt in his seminal work *Good Strategy, Bad Strategy*, good strategy is defined as "a coherent set of analyses, concepts, policies, arguments, and actions that respond to a high-stakes challenge." It is a plan for action backed by a cogent argument — an effective mixture of thought and implementation with a basic underlying

structure. Rumelt calls this the “kernel,” the bare bones center of a strategy — the “hard nut at the core of the concept.”[3]

A good strategy contains three essential elements:

> **Diagnosis**: The diagnosis explains the nature of the challenge, identifying which aspects of the situation are critical and the obstacles to be overcome. It answers the critical question, “What’s going on here?” Rumelt asserts that most deep strategic changes are brought about by a change in diagnosis — a shift in how the organization perceives its situation. In our context, this means understanding the technical and business challenges we face and recognizing the critical factors that will influence our success.
>
> **Guiding policy**: This is the overall approach for dealing with the challenge: a method to cope with or overcome the obstacles identified in the diagnosis. A good guiding policy defines a method of grappling with the situation and rules out a vast array of possible actions. Without a guiding policy, an organization would lack the principle to coordinate or focus its actions. For us, this involves setting a clear direction for our technical strategy that aligns with business priorities and long-term goals.
>
> **Coherent actions**: These are the specific steps required to carry out the guiding policy. The actions adopted should be consistent and coordinated. This means that our technical initiatives, from reducing technical debt to implementing new features, must be aligned and executed in a way that supports our overall strategy.

According to Rumelt, most strategies are "bad." Not because the strategy is wrong but because it is not a strategy at all. Often, what passes for strategy is merely a goal or, at best, a tactic. For instance, "reduce customer churn by 15%" is not a strategy. "Reduce customer churn by 15%" is a goal without context or a plan. It lacks the necessary diagnosis of the problem, a guiding policy to address the challenge, and coherent actions to achieve the desired outcome.

[3] Richard Rumelt, *Good Strategy, Bad Strategy: The Difference and Why It Matters* (Crown Business, 2011)

## Section 2

# Diagnosis

> *"You are an expert in your business domain, shaping technical investments to unlock long-term business capabilities."* Strategy facet
>
> *"You look outside Zillow—you know most problems are not new. You learn from the larger industry to improve and innovate by examining case studies, exploring new technologies, and importing best practices."* Learning facet

*— Zillow Leveling Guide*

Crafting an accurate diagnosis is essential. Fail to understand the problem correctly, and the strategy will probably result in wasted time and effort.

You may realize that you need to begin diagnosis for any number of reasons. Some will be obvious — a major product feature needs to be launched — others, less so — an observed trend across RCAs, a persistent sense that your organization is struggling to deliver features as fast as it ought to be, exposure to a new technical pattern suggesting significant gains in efficiency or adaptability, etc. Regardless of the initial trigger, remember that diagnosis is not the act of identifying solutions to a problem, it's the act of *defining the problem itself.*

To effectively diagnose a problem, you must have an accurate understanding of both the engineering and the business context. To understand the goals, constraints, and capabilities of only one but not the other is to invite misdiagnosis.

Two critical techniques we will focus on are **understanding business priorities** and **understanding engineering priorities**. These techniques are essential for the diagnosis phase of our strategy. Understanding business priorities involves understanding both the long-term vision and immediate goals of the business, and ensuring that our technical strategy supports these objectives. Conversely, understanding engineering priorities means identifying the technical challenges, such as technical debt, scalability, and maintainability, that must be addressed to ensure long-term success of the business.

For a principal engineer, the engineering priorities tend to be easier ones to build and maintain context on. Some of these are straightforward technical concerns, such as technical debt, long delivery cycles, or the strengths or weaknesses of a

particular architecture. (We discuss this step in more detail in *Understanding engineering priorities.)*

**Antipattern**

Don't accept a problem definition someone else has written at face value. Chances are, they haven't considered the same complete context that you have — the full sweep of technical and business constraints and opportunities. Always probe to understand the problem better. We are at our most effective when, by shifting the parameters of the problem, we resolve it with minimal — or even no — effort.

Understanding the business priorities tends to require more work. You must make an effort to understand where the business is going, how it maps to your organization, and how your technical choices will either support or conflict with those goals. In the end, this software exists for one reason: to power the business.

By effectively identifying and deeply understanding both business and engineering priorities, we can create a guiding policy that balances these needs, and develop coherent actions that drive our strategy forward.

The final step in establishing a diagnosis is to accurately weigh the many different problems you face and identify the critical ones — ideally, rooting several problems into a handful of more fundamental ones. (I.e., are the high operational load, frequent deadline misses, and low morale tied to an architectural pattern? A cultural problem? A poor division of responsibilities? Something else?) Prioritize the most important.

## **Technique:** Understanding business priorities

Understanding business priorities is really an act of inspection, discussion and understanding, which requires partnership with your customers, product partners and other stakeholders. Your primary goal in this process is to understand *why*. As a principal engineer, you should proactively seek to understand:

- **Key performance indicators (KPIs):** What are the critical metrics the business is focused on driving? How will the success of new features or initiatives be measured?
- **Motivating factors:** Beyond the feature request itself, what is the underlying business problem or opportunity it aims to address? Understanding the *why*

behind requests provides crucial context. Sometimes, understanding the *why* will lead you to completely different solutions from those that were initially requested or suggested.

**Antipattern**

In the pursuit of a robust technical strategy, it's crucial to avoid allowing business priorities to become the sole input into planning and strategy. While understanding and aligning with business goals is essential, a balanced approach that also considers engineering priorities is necessary for long-term success. A bad technical strategy, after all, will impede future business priorities.

- **Definition of success:** How does the business define a successful outcome for this project or initiative? This goes beyond technical completion to encompass the desired business impact. Taking this a step further — what would failure look like? Understanding where to draw the line can help identify a need to stop and reassess, or it may help identify key moments in the product development process which could be iteratively delivered to customers.
- **Customer needs and pain points:** Engage with product partners (and even customers) directly to understand their needs, preferences, and challenges. This direct insight is invaluable for aligning technical solutions with actual user demands. The more you do this, the more you will start to see similarities and patterns in what is being requested, helping you identify and design generalized solutions that may be able to satisfy many customers at once.Recently, new tools have emerged which are particularly powerful in understanding business context and helping you refine your own thinking; LLMs. LLMs are perfect tools for streamlining this technique of understanding business priorities. You have a powerful summarizer to pull key points from internal documents, you have as much time as you need to ask deeper questions, and you have a powerful researcher to assess your competition. LLMs are also excellent thought partners, helping you have a natural conversation to improve the depth of your understanding.

**Scenario**

A new key product initiative is taking shape. Leadership is referring to this initiative as the Sellers and Buyers In Zillow (SBIZ) program. Your senior product partner has presented an initial narrative that shows a new feature concept that will support virtual home tours. They have defined a high-level experience which allows a buyer to log in to Zillow and go on a virtual tour of a home. You realize this would be great to help with buyer schedules — you can tour homes whenever

it suits you, no need to coordinate access! Your team is being asked to help estimate what would be required to build this feature into the Zillow app, and whether they can deliver the first feature called "Connect with Virtual Agent" by the end of the quarter.

You reflect on how to respond to this request, and come up with a few options:

1. **Provide low-confidence estimates.** You already have a system which can create AI-based video-like renditions of a home based on the listing photos. It shouldn't be too hard to put this behind a 'Virtual tour' button on the website. You could give a reasonable ballpark on how big this project would likely be and start thinking about pulling this into quarterly planning.
2. **Ask for a product requirements document (PRD).** You don't like giving estimates without more information, and so you ask your product partner to help build a more thorough PRD which will ensure your estimates cover all expected requirements for this feature launch. You want to make sure you understand the scope — such as which web pages this should appear on, and how many photos we would require before we'd be able to create an AI-generation pipeline.
3. **Ask *why*.** The first thing you do is ask your product partner for more details on what they are trying to achieve. You want to understand what goals the business has and what critical business metrics we are looking to move the needle on. You also want to understand if this is a concept we have conviction on, or it's a narrow experiment to test buyer engagement, as that will affect the level of operational readiness you put into the feature.

**Antipattern**

Don't assume that your product partners have thought through everything. Ask questions, seek to understand and clarify both the goals and the approaches being suggested. Challenge assumptions and attempt to articulate your understanding of the problem domain to ensure you understand why we are looking to build out a particular strategy.

Thankfully, you chose option three, because after that first discussion with your PM partner, you realize you had made some really incorrect assumptions about what was intended here, and the UX mocks you saw really didn't line up with what you now understand the vision to be. Your product partners are looking at how to make it easier for buyers to tour homes on more flexible schedules, but

the goal was really to have buyers connect with a real estate agent who would physically tour the home and work remotely with a buyer over a video interface to explore the property on their behalf.

## **Technique:** Understanding engineering priorities

Business priorities inform and shape our strategy, but they need to be aligned with the technology landscape to be effective. Understanding and articulating engineering priorities helps us identify the technical challenges and opportunities that impact our ability to achieve business goals. This involves assessing factors like learning new technologies, managing technical debt, addressing system complexity, evaluating feasibility, leveraging existing solutions, and identifying related problems that could be solved by incorporating them into our strategy.

Engineering priorities are the foundation for executing business strategies. For example, adopting new technologies can keep us competitive and improve system performance. On the other hand, technical debt and system complexity can slow us down and need to be managed to avoid inefficiencies. Evaluating the feasibility of solutions ensures we pursue realistic goals.

Leveraging existing solutions where possible and innovating when necessary helps us balance proven technologies with new advancements. This approach ensures our technical strategy is sustainable and forward-looking. By understanding these priorities, we can align our technical capabilities with business objectives, making our strategy both realistic and ambitious.

In short, understanding engineering priorities allows us to accurately assess the technical landscape and create a strategy that aligns with business goals while being grounded in technical reality. This ensures our initiatives are feasible, sustainable, and capable of driving long-term success.

## **Technique:** Learning from the industry

Many of the problems faced in software development are not new. Nor are they unique to a single organization. The industry as a whole has encountered and addressed similar challenges to those you will

**Antipattern**

Don't just assume that because it worked elsewhere, it will work here. By the same token, don't fall into the trap of 'not invented here,' where the exact opposite problem can manifest — ignoring all industry guidance because it couldn't possibly work for your unique company.

encounter in your career, meaning a wealth of knowledge and best practices is available to draw from. By looking externally, we can leverage proven technologies, dependencies, and frameworks that have been successfully implemented elsewhere. This approach not only helps in solving current issues more efficiently, but also ensures that we are aligning with industry standards and innovations. Learning from industry case studies and best practices allows us to make informed strategic decisions, ultimately balancing the need for immediate solutions with long-term technical capability and business value. Thoughtful use of AI can make external research more efficient and focused — use it to support evaluations and surface relevant trade-offs.

**Scenario**

Now that you have a better understanding of what the business is trying to achieve with the virtual tours concept, you start to map out the technology landscape. You realize that we could perhaps enhance the new Zillow Messaging platform to include video calling as a capability, and you know there was a hack week project that showed how this could be done relatively simply with the existing architecture. You know that we need to account for buyers who are already working with an agent as well as those who are just starting out, so you do some research to understand how the agent lead pipeline works. Looking back at the last major feature your team launched on the Zillow app, you recognize that it took much longer than initial estimates — there were some architectural challenges that made it difficult to align work across multiple engineering team roadmaps, as well as some legacy code that has recently been causing crashes related to memory exhaustion on certain devices. This will all need to be considered as you give shape to your strategy.

After your initial technical diagnosis, you set yourself some next steps. First you'll meet with the app team leads to understand what challenges they are working through. Then, you'll do some online research to understand how competitors might be approaching similar problem spaces — and what video conferencing platforms might be leveraged to reduce the time-to-market and reduce the overall workload in building out and operating such a platform.

| Exercise | Thinking about your current project focus, reflect on whether and how you approached a diagnosis of the key problems you are attempting to solve. Did you deeply explore the business *and* engineering priorities? Was there data you were able to collect from industry research to inform you further? Was this an intentional phase of your planning and the development of your solution approach? What opportunities to look outside your company for inspiration or guidance did you take? |
|---|---|

**Section 3**

# Guiding policy

> *"Strategy is the act of substituting ambiguity for a set of options and making a reasoned choice among those options."* Strategy facet
>
> *"You influence and shape team architecture, working closely with Principal Engineers and other Seniors to design, adapt, and grow your architecture to respond to changing business needs."* Architecture facet
>
> *"The feedback cycles are longer than before, stretching across quarters and into years—a solution must not only deliver incremental value, but withstand the test of time via resiliency, scale, and adaptability."* Strategy facet
>
> — *Zillow Leveling Guide*

If accurately diagnosing the problem is the most critical part of authoring good strategy, arguably the hardest part in the engineering context is defining good guiding policy. As engineers, we tend to be good at figuring out what's wrong — especially when we understand that we should step back and consider the wider socio-technical and business contexts. We're also pretty good at coming up with coherent actions: we've been proposing and actuating concrete work since the start of our careers.

Let's return to the definition we offered above:

> A guiding policy is the overall approach for dealing with the challenge: a method to cope with or overcome the obstacles identified in the diagnosis. A good guiding policy defines a method of grappling with the situation and rules out a vast array of possible actions.

This is trickier: you need to define a principled approach. There is no easy way to generalize how to go about defining the *correct* or *best* approach — or what will theoretically work or won't. This, ultimately, rests on your shoulders. However, in this book we can offer this: in software engineering, a guiding policy will tend to boil down to defining an approach through one of three (or some combination of the three) factors:

1. **Technical architecture** — Architecture enables certain possibilities and constrains others. If we conceive of architecture not as a one concrete deliverable (a coherent action) but as deliberately enabling a set of cohesive possibilities, it becomes the basis for a guiding policy. Let's consider a quick example:
    - You diagnose that teams across the company are struggling to build an integrated product experience because of differing models for identifying a customer. A couple of potential guiding policies are identified:
        - **Build point-to-point integrations between different lines of business**. This may make sense if the business's long term goal isn't clear. Perhaps an integrated product experience is an experiment and may be abandoned, and this approach makes it easy to deliver the most critical integrations.
        - **One authentication authority will be used by all systems.** If the business's long term goal is clear — to build, say, a housing super app, then this offers a more sustainable and cohesive long-term approach.
2. **Organizational architecture —** Conway's Law states that technical architecture tends to follow organizational structure. We may realize that our primary problems are ones of coordination — e.g., building duplicative features, defining mismatched schemas or contracts, or creating disjointed solutions. In this case, the guiding policy may not be to build the architecture to meet the problem, but to work with management to reshuffle

the organization in a way that makes the problem disappear by aligning incentives. Let's consider a quick example:

    - You diagnose that a platform in your organization is struggling to meet the needs of the software development engineers who rely on it. It appears that the platform team is prioritizing features in a vacuum. Another team, under another director in your org, is creating a "user friendly" interface to this platform — this team more routinely meets with customers around the organization. As a result, a great deal of duplicative mapping architecture is being introduced, substantially increasing technical complexity and complicating both operational and feature work.

    - In this case, the best guiding policy may be to better align these two teams within the organization. Perhaps to make them sibling teams under the same leader; perhaps merge them entirely. The particulars can be considered as coherent actions — the guiding policy is to use an "inverse Conway maneuver" to align incentives and communication structures.

3. **Culture** — The attitudes, values, and behaviors of the collective individuals (engineers and managers) that constitute your organization will be a significant — even primary — determinant of outcomes. A guiding policy may be to change a particular cultural dimension — e.g., emphasize operational excellence, or create a welcoming environment for design inspection. Let's consider an example:.

    - You diagnose that code changes are not rigorously tested prior to release,and that a general attitude holds among junior engineers that values speed over quality. As a result, bugs are being introduced — and reintroduced — which impacts the customer experience and ultimately slows down long-term velocity.

    - You might decide that your guiding policy will be to swing the pendulum back from speed to caution. You communicate to leadership that you will be heavily focusing on quality and correctness with juniors in order to start a cultural course-correction. The particular coherent actions might look like insisting on written test plans and inspecting MRs for automation test cases.

Next we'll discuss approaches to consider when constructing a guiding policy.

## **Technique:** Delivering incremental value

> "Working software is the primary measure of progress."
>
> — *Agile Manifesto*[4]

The business will naturally bias towards delivering value to customers as early as possible. This is how we continue to keep customers engaged with our product, and as principal engineers, it's our job to pave the way towards the long-term vision by being intentional and thoughtful about the incremental steps we take. This means not only addressing immediate needs but also ensuring that each step aligns with and supports the broader strategic goals.

Successful teams have momentum, and maintaining this momentum requires a guiding policy that supports continuous value delivery without compromising quality or operational rigor. Delivering incremental value is a strategic approach that enables us to continuously and iteratively deliver value to customers while maintaining a clear focus on long-term goals. Unlike traditional methods that may involve large, infrequent releases, progressive delivery emphasizes incremental, controlled releases that allow teams to gather feedback, validate assumptions, and make adjustments in real-time. This approach ensures that each step taken is intentional and aligned with the broader strategic vision, rather than just a series of isolated forward movements.

This approach is fundamentally different from (and often mistaken for) simply *delivering frequently*. Without a strategic framework, incremental steps can lead to a fragmented and disjointed product that may meet short-term needs but fail to achieve long-term goals. *Incremental delivery*, on the other hand, ensures that each step is part of a cohesive plan, with clear milestones and checkpoints that guide the team towards the desired outcome. It allows for continuous improvement and refinement, ensuring that the product evolves in a way that is both a sustainable way of working and is aligned with the overall business strategy.

[4] Jeff Sutherland, Ken Schwaber, Kent Beck, et al., The Agile Manifesto, 2001.

**Antipattern**

It's critical to understand the difference between a strategic plan designed to deliver value incrementally and the common pressure engineering teams may feel to simply ship *something* to show value. These are diametrically opposed approaches to our craft, and only one of them is a thoughtful, intentional approach to balancing short term value and long term goals. Remember to always seek to understand business priorities, and you can avoid falling into the trap of date-driven development.

In part one, we discussed the technique of working backwards, which involves establishing a desirable vision and then determining the major system components needed to achieve it. That same methodology can be woven into your technical strategy. Working backwards helps us define *what* we want to achieve, and our technical strategy defines *how*. In the technical strategy we break our vision down into a coherent, logical roadmap, which aims to bring value to customers one step at a time. We ensure that the order of feature delivery makes sense, and that each release tells an evolving story to our customers that culminates in our desired end state. At any point in our journey, we may pivot or stop, knowing that even though we have not gotten to the end of our product roadmap as we initially intended, we have managed to deliver something of value along the way.  And by including engineering priorities in our planning, we have also left things better than we found them.

## **Technique:** Experimentation and learning

> *"You relentlessly seek to better understand the particulars of your dependencies, your customers, and how to influence others across your organization/the company."*
> Learning facet
>
> — *Zillow Leveling Guide*

In part one, we discussed building a proof of concept as a technique to help us experiment with and learn about technical concepts. It's a way to establish a clearer understanding of the feasibility of a particular concept or technology. There is another key technique that can be highly beneficial to drive learning: *learning from real customers*. Their feedback gives you direct insight into what they need, what's working, and where they struggle with your product. When you use those insights to shape your strategy, you build products that people actually want — leading to higher adoption and better results.

### A/B tests

**Antipattern**

A/B testing can be powerful, but it's not a strategy in and of itself. Relying on it for every decision can become an anti-pattern. Teams can lose direction, chasing small wins and risking optimizing for local maxima rather than long-term impact.

A/B testing helps you make smarter, data-driven decisions by comparing two or more different versions of a feature and seeing which performs better. It gives you empirical evidence for how users behave, allowing you to shape strategy early in the development process.

Run A/B tests early and with intention. Move quickly to gather your learnings, but be disciplined about what comes next. When tests fail, you must be rigorous in discarding the corresponding code, as it can contribute to long-term technical debt. Conversely, when a test succeeds, strengthen it. MVPs need to be hardened, ensuring that operational rigor and high quality are retrofitted before moving on. This approach helps maintain a balance between innovation and stability, allowing us to build robust systems that can evolve over time.

- **Move fast to learn.** Implement A/B tests quickly to gather insights and validate hypotheses. The faster we can learn from these tests, the sooner we can make informed decisions to improve our strategy.
- **Rigorously discard.** Be scrupulous about cleaning up failed experiments and their corresponding code. Keeping unnecessary code can lead to long-term technical debt, which can weigh the system down and complicate future development

## **Technique:** Develop a framework for prioritizing technical debt

> *"Try and leave this world a little better than you found it..."*
>
> — Robert Baden-Powell[5]

Technical debt happens when we choose expedient solutions over optimal ones — often to meet tight deadlines or deliver short-term business value. It's

[5] Attributed to Robert Baden-Powell, founder of The Boy Scouts Association, in his farewell address to Scouting, 1937.

sometimes unavoidable, but if it's not managed well it can lead to future difficulties in adapting to new and changing requirements, long-term inefficiencies, increased maintenance costs, and reduced system performance. A principal engineer's role is to help the product and engineering teams balance both tracks: moving fast when it makes sense, and investing in stability where it matters most.

**Identify and prioritize technical debt**

**Antipattern**

Simply labeling engineering issues as "tech debt" or "toil" is inadequate. It is crucial to clearly articulate the specific problems being observed and propose actionable solutions. This is especially important because not all partners are engineers and they need to be brought along in understanding the impact and necessary steps to resolve it. For example, instead of saying, "We have tech debt in our codebase," it is more effective to explain, "Our current architecture causes frequent outages and slow performance, which can be mitigated by refactoring our service layer."

The first step to paying off technical debt is identifying and prioritizing the areas of your codebase and architecture that need attention. This involves analyzing metrics, conducting code reviews, and gathering input from team members, including understanding challenges and concerns which your product and design partners are facing.

Maintain a *debt list* within your backlog. Each time you incur debt, enter the tasks needed to pay off that debt into your tracking system, along with an estimated effort and schedule. Use the debt backlog to track your tech debt progress. **Any unresolved debt more than 90 days old should be treated as critical.** Approach this kind of work in the same way as you do product asks. If you're using a scrum process, maintain the debt list as part of your product backlog. Treat each piece of identified debt as a scrum story, and estimate the effort and schedule for paying off each debt — the same way you estimate other stories in scrum. Prioritize debt based on its impact on functionality, maintainability, and business needs.

Work with your product partners to educate them on the consequences of each particular piece of debt: Is it impairing product quality or velocity? Is it leading to an increase in incidents which are detracting from the team's overall capacity and ability to devote time to other priorities? Is it resulting in a higher spend on infrastructure or tooling than we would like, impacting our bottom line? If so, work with them to regularly prioritize this work against other items in the team backlog, ensuring all participants in the prioritization conversation understand the relative tradeoffs for each decision. Remember: **We are all owners,** and work planning is a

shared exercise in which you, as a principal engineer, are expected to have a strong, well-informed and well-balanced voice.

**Guiding policies for mitigating technical debt**

Establishing guiding policies for mitigating technical debt can be a crucial step in ensuring that this type of work is consistently prioritized and managed effectively. Without a clear framework, technical debt can accumulate unnoticed, leading to increased maintenance costs, reduced system performance, and slower delivery times. By implementing these policies, teams can align their technical initiatives with business priorities, ensuring that both immediate and long-term goals are met.

Below, we describe some established guiding policies which you may have come across in your career. Following this section we'll explore the tradeoffs between some of these ideas in our example scenario.

1. **Reserve capacity:** Some teams reserve a portion of their capacity for technical debt work, or they create dedicated projects for significant architectural initiatives. At Zillow we use the concept of *15% time*, which is intended to create space in both budgeting and planning so teams can focus on tech debt mitigation and other engineering-excellence work.
2. **Systematic overestimation:** A policy of systematic overestimation involves intentionally overestimating the time, resources, or effort required for tasks and projects. This approach can help create a buffer for unforeseen issues that reduces the risk that the engineering team will be left with inadequate time to deliver high-quality systems and prevent the accumulation of future tech debt.
3. **Feature flags governance:** You have likely fallen victim over time to feature flag bloat and poor practices around their implementation. To mitigate this, you may consider implementing guidelines for the *use of* and *removal of* feature flags across your codebase.
4. **Service level objectives**: SLOs can track the reliability of services. By setting clear SLOs, teams can monitor and improve the operational excellence of their services. This ensures that reliability and performance are consistently measured and maintained, reducing the risk of accumulating technical debt due to neglected service quality.

5. **Code coverage metrics:** *Code coverage* is a metric that helps you understand how much of your source code is tested. It provides insights into line coverage, conditional coverage, branches coverage, and functional coverage, which can be used to improve the quality and reliability of the codebase. Establishing minimum expectations for the level of test coverage on new feature work, for example, may be a way to limit the accumulation of a particular kind of technical debt and ensure that quality and testing are embedded into work planning.
6. **Test planning:** This involves creating detailed test plans that cover various quality areas, including functional, non-functional, and exploratory testing. This work should be performed early in the development cycle in partnership with your PM to ensure that test plans are part of the requirements definition process. This allows you to identify and mitigate risks early, improve code quality, and maintain a sustainable development pace.
7. **Regular operational reviews:** Operational reviews involve collecting and analyzing key engineering and operational metrics across the organization to ensure processes, coding activities, and resource allocation are optimized for high performance, reliability, and quality. When both engineering and product teams have visibility into this data, you can better understand how your software operates in practice, respond to incidents, anticipate future problems, and proactively build more resilient systems.
8. **Root cause analysis reviews:** RCA reviews involve a thorough examination of incidents to identify the underlying causes of customer- and business-impacting incidents, and to implement corrective actions to prevent recurrence. By conducting regular RCA reviews with both engineering and product teams in the room, you can ensure that lessons learned from past incidents are documented and shared across teams, promoting a culture of continuous improvement and accountability.

In each of the following scenarios, you could apply a guiding policy to mitigate technical debt or reduce the risk of technical debt accumulation. Let's evaluate the risks and tradeoffs associated with each policy, then select the best option for each situation.

**Scenario 1**

You've developed a clearer understanding of what the business wants to achieve with the Virtual Tours concept. You've mapped out the technology landscape through internal discussion and analysis, as well as external research, to inform your technical strategy. You now have a reasonably clear plan for how to build a robust video conferencing solution that will enable buyers to connect with an agent in the Zillow app, then schedule and "attend a virtual walkthrough of a property. You work with several teams to craft a high-level architecture, and you work with the program TPM to produce an initial, medium-confidence plan that would let you deliver this product in around six months. You bring this timeline to your product partners, who push back, indicating that there is a significant marketing opportunity to be leveraged if we can ship an MVP of this product by the end of the current quarter.

Considering that the business priority has been identified, what guiding policies could you include in your technical strategy to mitigate the accumulation of technical debt and project risk while still moving up the launch date to take advantage of the marketing opportunity?

1. **Reserve capacity**

   While this can be effective for getting technical debt work prioritized, it should be considered a method of last resort. Reserving capacity is a defensive approach that lets teams address *non-specific* tech debt during work planning. A more integrated approach to avoiding new tech debt would intentionally manage technical and feature work within a single backlog. In this scenario, reserving capacity may help reserve time for operational excellence and tech debt mitigation work, but it's also likely to make it harder to meet the marketing date goal. Reserving capacity may also lead to a corresponding scope reduction across the product, which could impair the overall customer experience.

2. **Systematic overestimation**

   This approach can create a buffer for unforeseen issues, improve planning and resource allocation, enhance team morale and productivity, and build stakeholder confidence. However, it also has potential negative consequences. Overestimation can lead to inefficiencies and wasted

resources if the buffer is consistently too large. It may also result in complacency, where teams don't strive for optimal velocity, and it can create a false sense of security among stakeholders. This policy may also result in lost trust with stakeholders and partners, which can be very difficult to correct. In this scenario, it's also likely to make it harder to meet the marketing date goal, which could result in a corresponding scope reduction across the product and an impaired overall customer experience.

3. **Service level objectives (SLOs)**

   Setting these goals during product development gives teams agreed-upon expectations for quality outcomes and limiting the accumulation of tech debt. The downside of this policy is that you may not be able to tell whether you've met your goals until *after* you ship the new work. You may need to add specifics, such as establishing minimum expectations for the level of logging, monitoring and alerting that should be implemented with all software changes. In this scenario, it would be beneficial to align early with your PM partners on quality and success metrics so you can have collaborative and informed discussions about work prioritization and risk management throughout the project lifecycle.

Option 3 is generally the best approach, as it aligns both engineering and product stakeholders to a common goal with measurable results. The first two options — though pragmatic and often used in the industry — reduce communication and, in the case of Option 2, can breed systemic distrust between stakeholders.

### Scenario 2

During development of the new Virtual Tours product, there is a production incident with one of the key services being refactored: the tour scheduling service stops responding to requests from the front end. Customers who attempt to schedule in-person tours are met with a generic 404 error page. The issue lasts for several hours and the team eventually root causes the problem to a recent deployment which didn't leverage a feature flag as the implementation plan called for. Your manager schedules a 1:1 with you to discuss the incident and explore ways to prevent a recurrence of this sort of problem during the remainder of the Virtual Tours project. Looking back, what guiding policies could you have introduced or improved to mitigate this sort of risk to the project?

1. **Feature flags governance**

Establishing mechanisms and processes to remove unused feature flags (and the corresponding code) over time can prevent or reduce software complexity and the accumulation of technical debt. Feature flags are essential for experimentation and coordinating software releases, but they should *not* replace software testing. In this instance, the team is likely using feature flags incorrectly as a protection mechanism and not implementing the right level of testing for the system(s) they are changing. Feature flag governance is a good policy to apply to address existing technical debt, but a stronger mechanism here might be to focus on improving the team's testing and verification practices.

2. **Code coverage metrics**

   Code coverage alone isn't a reliable measure of test quality — but it's still useful alongside other methods.  Over-focusing on the metric can encourage poor habits that only improve the one number rather than verifying other complex, critical  areas of the system. In this situation, constructing a more intentional and thorough test plan — as well as establishing some baseline code coverage expectations — may strike a good balance.

3. **Test planning**

   This project may have been missing early PM alignment on the scope and approach to testing. Setting these expectations early during product and UX design will help you have a more collaborative and informed discussion around work prioritization and risk management throughout the project lifecycle. Establishing a test plan upfront can set expectations for the need to continuously introduce automation testing for all code changes throughout the project, as opposed to relying solely on feature flags for protection.

4. **Root cause analysis reviews**

   This practice can help identify systemic issues and areas of technical debt that need attention, thereby improving the overall quality and reliability of the software systems. Like operational reviews, RCA reviews can be a powerful tool to drive remediation actions, but will largely drive outcomes as a reactive *response* to negative observations or incidents rather than as a proactive *prevention* measure. In this instance, the team should definitely

perform an RCA review for the incident, but this policy is by design a retrospective, and will not itself reduce ongoing risks.

None of the above options are a silver bullet — such is the reality of engineering complex systems. Option 4 might be the best approach for this particular situation: the problem remains ambiguous — we're not quite sure why the feature flag wasn't leveraged as planned — and so running a retrospective process with the engineering group is a necessary first step to understand the problem itself. Once we know more, we'll be better positioned to evaluate the relevancy of the other options to prevent the issue from recurring in the future.

Ultimately, selecting guiding policies to apply within your technical strategy is an exercise in evaluation risks and tradeoffs. No system is ever perfect, and maintaining a balance between product innovation and technical excellence is an evergreen challenge.

| Exercise | What guardrails or tenets could you put in place to help balance the need to move fast ( business priorities), with the equally important need to deliver high quality products that move us towards the technical vision (engineering priorities)? Reflect on the discussion of technical debt mitigation:  What guiding policies have you applied or attempted within your organization to address tech debt within systems you own? What has worked? What hasn't? What other ideas do you have for guiding policies that you could apply to your current project? |
|---|---|

**Section 4**

# Coherent actions

> *"You understand that a curated set of fundamental strengths enables a great variety of particular features, quickly and affordably. You build systems that anticipate change; you enable an architecture where it is easy to disband and replace particular systems when the need arises."* Architecture facet

> *"Furthermore, you treat operations as an architectural concern, treating operational health as a first-order design requirement."* Operations facet
>
> — *Zillow Leveling Guide*

Coherent action is the third — and arguably most important — element of a good strategy. As mentioned earlier, a strategy without actions is not a strategy at all. Too often, leadership teams consider high-level objectives their strategy and stop there. But a good strategy must contain action: without it, strategy without action is like an unsharpened knife. It looks useful, but it can't do the job.

Let's revisit our description of coherent actions:

> **Coherent actions** are the specific steps required to carry out the guiding policy. The actions adopted should be consistent and coordinated. This means that our technical initiatives, from reducing technical debt to implementing new features, must be aligned and executed in a way that supports our overall strategy.

In many respects, defining coherent actions is the easy part of technical strategy — not because the details are simple or the methods straightforward but because this is what we've practiced all of our careers. Define a project plan to deliver a feature; identify a model to refactor roles and responsibilities; find the scaling bottleneck; so on. You are an expert in your domain, and you probably have a pretty good sense of how to go about taking specific concrete actions in accordance with your guiding policy. Coherent actions are the sharp end of our strategy.

So this section will speak to different patterns for approaching specific problems. These are tried-and-true ways of developing coherent actions for different classes of problems.

## **Example:** Strangler fig

As engineers, we are often faced with a legacy system — typically a monolithic architecture — which requires refactoring and perhaps re-platforming in order to accommodate a future set of planned capabilities. It can be difficult to plan how and when to do this type of work. It is usually a large undertaking, and a risky one — we don't want to break the product experience, but we must be able to improve and modernize our system architectures and infrastructure dependencies.

The strangler fig approach is a method for rearchitecting a system incrementally, allowing for continuous operation while gradually replacing parts of the old system with new components. This technique is inspired by the way a strangler fig plant grows around a host tree, eventually replacing it entirely.[6] The approach involves building new functionalities alongside the existing system and slowly migrating users to the new system. Over time, the old system is phased out as the new system takes over. This method minimizes disruption and allows for iterative progress, making it a practical solution for complex system migrations.[7]

**Antipattern**

Relying on an assessment of operational or business metrics to verify the correctness of a refactored solution can't replace thorough system testing. Refactoring and re-architecture projects must have a cohesive and thorough test plan, ideally driven through automated regression testing — the most reliable form of quality assurance engineers have access to.

## **Example:** Shadow runner

Refactoring and re-architecting a system often presents the highest degree of risk in our work as engineers. We already have customers, and they have established expectations. It's imperative that when performing a like-for-like refactor of an established system, we are able to verify that the changes we make leave the system functioning as it did prior to the refactor. In the industry it's often said that unit testing is a useful tool for this purpose, but the reality is that unit tests are far more tightly coupled to the implementation than they ideally should be. They may cover fewer of the critical parts of the codebase than we would like, making unit tests deficient as a reasonable measure of quality assurance.

The shadow runner approach is a method for testing a system by running a parallel version alongside the production environment. This technique allows developers to test non-functional changes using a subset of production traffic without impacting the live system. The shadow runner platform captures and analyzes production traffic to identify regressions in responses and latency. By

[6] Martin Fowler, "Strangler Fig Application," MartinFowler.com, August 2024.
[7] "Strangler Fig Pattern," Microsoft Learn, Azure Architecture Center, February 2025, https://learn.microsoft.com/en-us/azure/architecture/patterns/strangler-fig.

deploying changes in a non-invasive manner, it enables teams to collect empirical data over time, surfacing small deviations that might otherwise go unnoticed. This approach is particularly useful for ensuring the stability and performance of critical systems during migrations or updates.

### **Example:** Stitch a solution together with steel threads

For large projects, craft a narrow use case that spans the systems under development and demonstrates real business value early. This technique is sometimes called a *steel thread*. *Thread* refers to the use case's narrow focus: just enough to prove value while spanning the relevant architecture, ignoring edge cases in favor of the primary path of execution. . *Steel* points to the durability of the technique.. By cutting end-to-end through a fundamental use case, it creates an architecture others can build on. .

The steel thread approach has a number of advantages:

- Enables teams to work in parallel as soon as the steel thread is built, lowering team dependency
- Derisks the bulk of the complexity early by tackling an important edge case first, as opposed to nibbling on the edges of a problem
- Scores an early win that can build momentum and motivate future development and/or identify fundamental challenges early in the product life cycle
- Forces collaboration by focusing attention on the points of integration between the various systems. Integration points are often the most challenging to change later because they typically involve coordination across many teams.
- Breaks down a large, complicated project into a more manageable first step that can be iterated further

A steel thread is in many ways the architectural equivalent of a proof of concept, which is defined at the implementation level. They both aim to quickly demonstrate feasibility and business value early. They are both effective at defining a manageable initial step in the context of a larger project by ignoring details that are extraneous to the primary business objective. The major difference

is that a proof of concept is typically intended to be abandoned after implementation, whereas the steel thread is meant to create an enduring architectural pathway to iterate upon. In this way, the steel thread requires more planning and collaboration than a proof of concept, which may be a solo effort.

**Antipattern**

A steel thread is often confused with a prototype. While a prototype is typically a throwaway model used to test concepts, a steel thread is meant to be an enduring architectural pathway that should be iterated upon. Treating a steel thread as a prototype risks neglecting the necessary planning and collaboration required for a robust and scalable architecture.

The technique also shares some similarities with the concept of a *minimum viable product* (MVP), in that they both aim to quickly verify business value. A steel thread can be an excellent first step in building an MVP. However, a steel thread isn't expected to handle all the use cases and particularly edge cases that even an MVP might. A steel thread needn't be end-consumer facing at all — it merely needs to demonstrate the feasibility of the architecture in addressing the business case. You might also have noticed the similarities with the strangler fig pattern: an approach which is, in effect, a steel thread applied to a refactoring or migration project.[8]

**Exercise**

Do any of the examples of coherent actions outlined in this section resonate with you? Have you ever intentionally applied one of these approaches in your strategy? Can you see opportunities to use one of these actions as part of a strategy you are actively developing?

[8] If you'd like to read more about the steel threads technique, this short article is a great primer.

**Section 5**

# Putting it all together: Crafting a strategic proposal

> *"You are expected to identify, define, socialize, and break down novel problems in your organizational domain."* Strategy facet
>
> *"Your design documents are models of reasoned decision making, persuading the reader to accept a conclusion through the careful specification of the problem, thorough presentation of context, and rigorous analysis of trade-offs. You use data to bring clarity to contentious issues."* Communication facet

*— Zillow Leveling Guide*

In part one, we discussed the need to delve into technical uncertainties and proactively identify and articulate emergent problems. As a Principal Engineer, your role involves reducing ambiguity by crafting coherent plans and architectures for new business initiatives, assessing whether solutions are addressing the problem correctly, and persuading others to understand and address these issues effectively.

We also discussed a couple of techniques to help you drive down ambiguity: proof-of-concepts and two-way doors. In this section we will introduce you to a few more techniques to help you craft a comprehensive and thoughtful strategy that marries business priorities and technical needs.

## **Technique:** The power of writing it all down

We've discussed the significance of writing and communication as essential tools for influencing and guiding technical projects. And we've stressed how articulate and timely communication helps prevent misunderstandings, keeps projects on track, and fosters a collaborative work environment.

Now we'll discuss how to articulate your strategy clearly so that it  helps you build consensus, and drive towards the outcomes you want..

**Assume zero context**
Assume that those reading your document may not be familiar with all other documents and prior discussions. When articulating your strategy, be sure to summarize the diagnosis you have constructed and the guiding policies that are influencing how the strategy is shaping up.

**Why before what**
Much more important than *what* you are proposing is *why* you are proposing it. This gets to the heart of the diagnosis of your strategy, and this context is crucial in helping others understand your thought process and decision-making journey. It also helps others reason about your approach, question assumptions you have made, and potentially offer alternative views that could achieve the same outcomes in a different way.

## **Technique:** Articulate risks and tradeoffs

Articulating risks and tradeoffs is a crucial technique in developing a robust technical strategy. This process involves clearly documenting potential risks and the tradeoffs associated with different strategic decisions. Doing so ensures transparency, aids in making informed decisions, and helps you gain stakeholder trust.

A comprehensive strategy document should include sections that detail the business priorities and goals that are driving your strategy and influencing the guiding policies. This might include a problem definition, scope details, high level architectural considerations and recommendations, and alternative options considered. Within these sections, explicitly outline the risks associated with each option and the tradeoffs involved if that option were chosen. This might include potential impacts on performance, scalability, maintainability, and cost. It might discuss how one solution leads to larger upfront investment for longer term benefits, or another approach will let us get faster learnings but also be less suitable to address our best understanding of the long-term objectives and lead to rework in the future. By presenting this information clearly, stakeholders can better understand the implications of each decision and the recommendation you are making.

In some cases, you may not yet have enough data to fully understand all risks and tradeoffs. There may still be key elements of uncertainty in your strategy, and that's where the following techniques can come into play.

## **Technique:** Strategic evolution

> *"Everybody has a plan until they get hit."*
>
> *— Mike Tyson*[9]

Technology changes. Business changes. A robust technical strategy is not static: it should be designed to actively collect and adapt to new information. We discussed particular techniques to **reduce ambiguity** through learning, such as , two-way doors, and quantifying problems. A good guiding policy should encourage their use.

We've also talked about the importance of establishing a **clear diagnosis** that considers both the business priorities you're aiming to meet and the technical landscape in which you're operating.

We also outlined a few **coherent actions** you can weave into your strategy, which might help you address particular classes of problems.

The final thought we want to leave you with is this: Be ready to embrace change, but don't shy away from building an ambitious vision for the future and building a plan to get there. This, fundamentally, is the mission of your technical strategy.

A comprehensive strategy should always encompass three key facets:

1. **Target state: Where we want to go —** This is the desired future state of the software ecosystem, aiming for high reliability, efficiency, and maintainability. It sets the vision and long-term goals that will guide all strategic decisions in the future, until the target state itself shifts.
2. **Current state: Where are we now —** This involves a thorough assessment of the current situation, identifying existing challenges such as technical

[9] The champion boxer has delivered versions of this quote numerous times over the years, as he explained in a 2012 article: Mike Bernadino, "Mike Tyson explains one of his most famous quotes," *South Florida Sun Sentinel*, November 9, 2012. Attribution via Quote Investigator.

debt, complex architectures, and inconsistent service reliability. Understanding the current state is crucial for diagnosing issues and setting a realistic path forward. Establishing mechanisms — such as a debt backlog or operational review processes — ensures that you and your team can maintain a relatively fresh understanding of the current State, as it continuously changes.

3. **Next steps: What we are doing next —** These are the immediate actions and coherent actions required to transition from the current state towards the target state. It's important that you recognise you may not ever reach the target state you established in this strategy, but being clear about the ultimate goal of your strategy ensures that every step you take is intentionally aimed towards that ambitious vision you have.

A few final methods to consider:

1. **Living documents**: Treat strategy documents as living documents that are regularly updated to reflect new insights, decisions, and changes in direction. This approach ensures that the strategy remains relevant and actionable over time.
2. **Continuous feedback loops**: Establish continuous feedback loops to gather input from various stakeholders, including engineers, product managers, and business leaders. Regularly review and incorporate this feedback to refine the strategy.
3. **Abandoning failed ideas**: Be willing to intentionally abandon ideas and experiments that do not yield the desired results. Recognizing and discarding unsuccessful approaches is crucial for focusing resources on more promising initiatives.
4. **Balancing innovation with established practices:** While it's important to adopt new technologies and methodologies to stay competitive and improve overall system quality, it's equally important to maintain stability and reliability for your customers by leveraging familiar, proven practices. This balance ensures that innovation does not come at the cost of operational excellence, and that the organization can adapt to new information while maintaining a solid foundation.

By incorporating these principles, you can ensure that your technical strategy is not only robust but also adaptable, allowing you to navigate the complexities of

the evolving technological landscape effectively, while maintaining a long-term focus on the future vision for the business.

| Final Exercise | Consider an upcoming project that you are involved in, where the development strategy is still in the early phases of development. Using what you've learned, create a 1-2 page strategy briefing for that project.<br><br>1. Diagnosis<br>  a. What is the primary business motivation that we're aiming to solve?<br>  b. What are the key engineering priorities that may influence our strategy for this project?<br>  c. Are there relevant areas of industry that could inform how we approach this project?<br>2. Guiding policies<br>  a. Outline 2-3 key guiding policies that will influence how your strategy will be executed.<br>3. Coherent actions<br>  a. Outline 2-3 coherent actions which you can take to advance the strategy. |
|---|---|

# Conclusion

You've reached the end — well done! We know you've invested time in this book, and we're confident you'll find yourself applying these skills in your career. You now have a toolkit to help you:

- Create outsized impact through leverage
- Cultivate the relationships that hold organizations together
- Leverage your influence without formal authority
- Scale your own practice sustainably
- Craft resilient technical strategies that serve both today and tomorrow

This book reflects our own experiences and challenges at Zillow — but the lessons apply just as well to any engineering organization that values deep technical leadership. If you'd like to dive deeper into our thinking and stories, check out the Zillow Engineering Blog, where we publish posts on architecture, best practices, and the projects driving our business forward.

We're always looking for brilliant engineers to join us. If you're passionate about building products at scale and want to help shape the future of real estate technology, explore opportunities on our careers page.

Finally, stay connected! Follow Zillow on LinkedIn to get updates on the business, conferences, talks, and community events.

Here's to your continued growth as a technical leader. May these practices serve you well — wherever your path takes you.

**— The Multiplier Authors**

# About the authors

This book was co-authored by four of Zillow's Senior Principal Engineers, each contributing equally to writing, editing, and refining the material. Together, they developed and co-led in-person training courses to reinforce the content of this book. They are listed below alphabetically by surname.

**Nathan Figueroa** is a Senior Principal Engineer at Zillow with over 20 years of software engineering experience, including 12 years at Zillow. He has led numerous impactful initiatives at Zillow, including designing systems for ingesting and serving property photos, migrating critical listing data workflows to streaming architectures, and significantly shaping Zillow's real estate data infrastructure. Passionate about fostering technical leadership and career growth, Nathan has been active in shaping Zillow's engineering career frameworks and interviewing practices. He lives in Seattle, Washington, with his husband and their four children.

**Kerry Hart** is a Senior Principal Engineer at Zillow, where he leads engineering strategy and architecture for the company's core platforms — including compute infrastructure, identity, communications, financial systems, and data architecture. Kerry has played key roles in shaping the company's data mesh vision, engineering role framework, and platform modernization efforts. Prior to Zillow, Kerry spent seven years at Amazon Web Services (AWS), where he designed, built, and operated next-generation infrastructure for core billing systems. He currently lives with his wife on a hill in New Hampshire, where nature's idyllic panorama is interrupted by the sound of their children making mischief.

**Ryan Lohan** is a Senior Principal Engineer at Zillow with over 20 years of experience leading complex, large-scale systems. A passionate software engineer and change leader, he is dedicated to helping organizations achieve engineering and operational excellence through sustainable evolution and durable mechanisms. Previously, he was a Principal Engineer at Amazon, where he played pivotal roles in AWS and the launch of Buy with Prime. Proudly Australian, Ryan brings a global perspective to his work, and has also been a certified PADI scuba

instructor. He now lives in the Pacific Northwest with his wife and three children, who keep him grounded, inspired, and frequently outnumbered.

**Ralph McNeal** is a Senior Principal Engineer at Zillow, shaping the design, architecture, and strategy for teams that deliver core services and platforms across Data & Analytics, Networking, Infrastructure, Operations, and DevEx. Prior to Zillow, he was a Staff Engineer at Twitter, responsible for building and operating distributed infrastructure that underpinned the company's enterprise-wide code repository, CI pipelines, and developer tooling. He also brings product-development expertise, having designed and delivered large-scale e-commerce platforms and globally deployed SaaS solutions. A native of St. Louis, Missouri, who began his career in high school, Ralph offers decades of hands-on experience leading high-impact engineering teams.

# Acknowledgments

This book would not have been possible without the many people who shaped its content, guided its direction, and helped bring it to life.

We are especially grateful to Kristin Acker for her steady leadership and unwavering support. Her belief in the importance of this material — and her behind-the-scenes work to clear roadblocks — enabled us to build something ambitious and lasting.

We owe deep thanks to the Learning and Development team for their thoughtful leadership in translating our materials into an engaging curriculum. From conducting interviews with principal engineers and synthesizing key insights, to shaping the course content from initial concept through final delivery, their work was foundational to both the training and this book. Their partnership throughout has been invaluable.

Many engineers and engineering managers across Zillow Group invested their time to review early drafts of the training materials and offer feedback that made them better. Their thoughtfulness helped ensure that what we built was both practical and grounded in the realities of our work.

Likewise, the Software Development Engineer Leveling Guide included in this book's appendix reflects the careful work of many contributors over the years. Engineers and managers collaborated to ensure the guide accurately captures the expectations and realities of technical leadership at Zillow.

And of course, thank you to our families and spouses, who supported us through the long hours of writing, editing, and iteration. Your patience and encouragement made this possible.

# Appendix

# Zillow Group Core Values

We place a huge emphasis on fostering a culture that encourages people to share their ideas and be able to access the support they need to turn those ideas into innovations that drive our company forward.

### Consumers are our North Star

With quality at the center, everything we build and deliver is with consumers in mind. We also partner with industry professionals to create the best consumer experience.

### Do the Right Thing

We operate with integrity in our own work, putting the good of consumers, our colleagues, business and the industry above individual interests.

### Turn on the Lights

Transparency is at the core of everything we do, making the implicit explicit by documenting, educating and sharing the why of our work. This ensures no one is left in the dark and everyone has the information they need to be empowered and effective.

### Own It

We aim high, move fast and deliver — each individual and team is accountable for our growth and impact from start to finish.

### Better Together

We navigate complex work, empower diverse perspectives, speak up and move forward together. We collaborate with purpose and trust by including the right voices.

### We Play to Win

We do the hard work, take bold chances and lead with excellence to help more people get home.

# Zillow Group Software Development Engineer Leveling Guide

## 1 Overview

This guide is intended to help you understand the Software Development Engineer (SDE) role here at Zillow Group. You are critical to the success of this company: it is you that authors the software and builds the systems that guide our customers through their real estate journey. This guide is intended to help you through a similar journey—your career.

A note on the philosophy that underpins this guide: careers are complex journeys, and they rarely follow a prescribed path. This guide seeks to capture the core qualities of the role at each level without defining specific gates or checklists. It is intended to foster richer career development discussions between engineers and management, not supplant them with an overly prescriptive model.

## 2 Role Description

As a Software Development Engineer, you are responsible for developing the software that underpins the products and services we provide. You design, author, operate, and maintain this software. You are accountable for the quality of the software you produce—the code and architecture you produce must deliver customer value while being secure, reliable, efficient, and maintainable. You are pragmatic, operating within resource constraints to build solutions that meet the needs of the business—and you do so at the speed of the business. You are a steward of our technical capabilities, anticipating future needs by building flexible, adaptable solutions.

The Software Development Engineer track is a long one. At junior-most levels, your impact may be measured at the feature level and be achieved through small code changes; at senior-most levels, your impact may be measured at the company scale and achieved through strategic initiatives. Across all levels, one constant remains true: you execute within the context of the organizational hierarchy—your team, your org; however, the scale you must consider tends to be at least one level above. A Senior Engineer may execute within their immediate team, but think about their sister teams; Principals (and above) are rooted in the technical context of their organization to ensure internal alignment, while considering how their organization impacts the broader company.

Therefore, software engineering is a fractal: the behaviors embodied at the task level recur at the team level, the cross-team level, the organizational level, and finally at the company itself. At junior levels, you may be encapsulating complexity by organizing code into functions or classes; at more senior levels, you may decompose a monolithic service into microservices; at the most senior levels, you're reducing organizational complexity by shaping communication structures or organizational hierarchies. They're conceptually similar but exist at different levels of abstraction and scope—a continuum exists across the role. The difference between an SDE-I and a Distinguished Engineer may be perceived as a gulf, but not as wide as it first appears.

## 3 Facets of the Software Development Engineer role

The Software Development Engineer role changes significantly over the full range of its ladder. The following seven facets are intended to get at the very root of the role: these are the things that, though they change in meaning and emphasis, are present in some way at each level. This guide is centered around them.

**Code.** Code is the foundation of your craft. You author code to solve problems for the business. You leverage intelligent tools and AI agents to optimize the coding process; you understand and own the results. You organize that code to be understandable by others and adaptable to changing needs, following the principle 'leaving it better than you found it.' You test the code to ensure it is correct and efficient. At the junior-most levels, the authoring of high-quality code will be your primary preoccupation. As you become more senior, you must not lose touch with the codebase, even as other concerns compete for your time and attention.

**Architecture**. The definition and arrangement of our software is foundational to our ability to adapt, scale, and operate our business. Architecture will have different meanings in different contexts, ranging from the organization of large codebases and complex libraries to the roles, responsibilities, and behaviors of distributed systems. At more junior levels, you will arrange your software into structures that are adaptable and maintainable in order to deliver specific features and behaviors. As you move up, your focus will shift to modeling complex domains in elegant solutions—systems that can be recomposed in response to an evolving business and scale to meet its needs. At senior-most levels, you treat architecture and organization as inseparable concerns—you are as focused on the human as on the purely technical.

**Operations.** You operate the software you author, and own its operations. You rigorously validate that you meet the contracts you've specified through metrics, dashboards, and alarms. When something breaks, you know before our customers, and you fix the issue with deep urgency. At junior levels, you will operate runbooks, debug minor issues, and learn the rhythms of how to operate software. As you grow, you will become the expert responsible for resolving the most pernicious and difficult issues. At senior-most levels, you cultivate operational excellence across teams and organizations, often without the luxury of leading by example.

**Communication.** Your ability to influence is a direct function of your ability to write about and speak to your ideas. At junior levels, this means authoring effective documentation and speaking to technical context. As you grow, you will come to communicate complex technical designs in written and verbal mediums. At senior-most levels, you will distill complex subjects for larger and more varied groups, tailoring your communications to audiences ranging from junior developers to executives.

**Learning.** As a software engineer, you never stop honing your craft and better understanding your domain. Change is the only constant in our industry, and you change with it. The act of

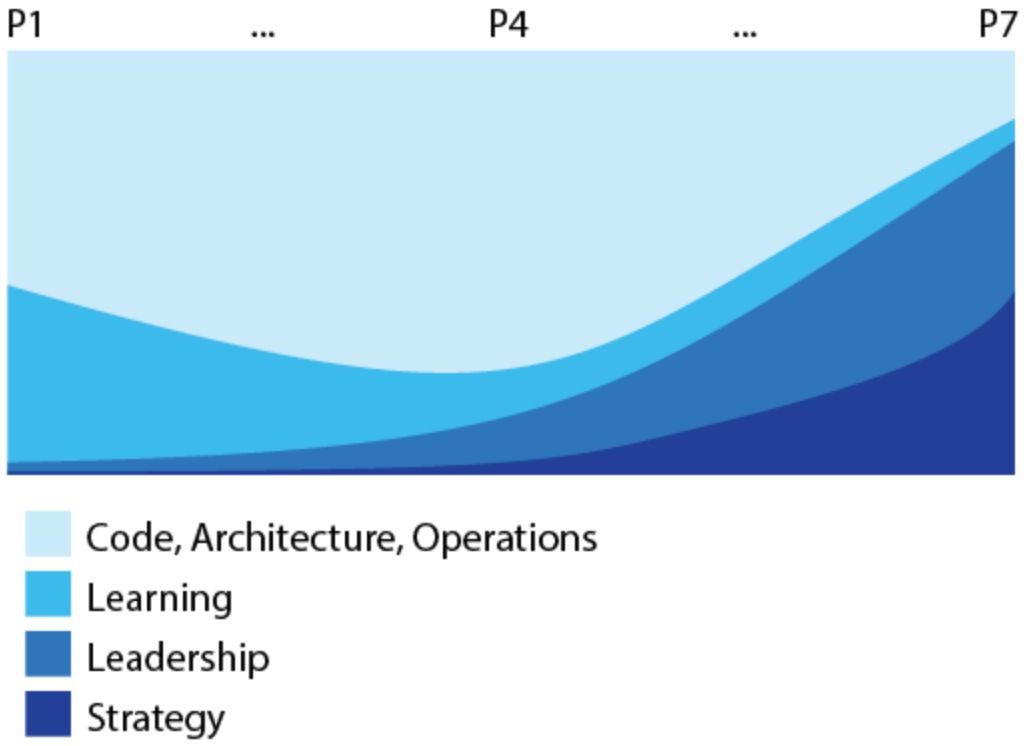
learning is core to the early levels (P1, P2, P3): you are expected to grow out of these and into P4. At these early levels, you are learning the craft skills (code, architecture, operations) necessary to advance. At P4 and above, the nature of learning changes. Advancing to the next level is no longer expected; you seek to improve or develop skills to better succeed in your role. You proactively seek out the latest tooling, try new methodologies, and adapt to new patterns of work. You are impatiently curious, quick to explore the ever-evolving technical landscape for better answers to enduring problems. You continue to refine your expertise in your business domain, seeking to ever-better align technology to business opportunity.

**Leadership.** You lead by influencing others without authority. You shape others through the information you share, the individuals you mentor, the issues you escalate, and the ideas you propagate. At junior levels, your leadership will be localized to your team: you will lead juniors to produce high quality output on time, and you will influence management regarding specific projects and solutions. As you move up, your leadership will expand beyond your team to your larger organization. You will guide other technical leaders to common cross-cutting solutions or shared architecture, and you will advise senior leaders and executives on matters of architecture, strategy, and organization.

**Strategy.** Strategy is the act of substituting ambiguity for a set of options and making a reasoned choice among those options. Your effectiveness at crafting strategy will be underpinned by your growing domain expertise (both technical and business). As you grow, the problems you solve will have a greater impact, the problem statement will grow more ambiguous, the trade-offs more difficult to identify, and the feedback loop on whether you made the correct choice will lengthen. At junior levels, you are seeking solutions to clearly scoped problems where the trade-offs among options are well understood by your seniors and the feedback loop is short. At more senior levels, your judgment and accumulated experience becomes more critical: you must identify and choose a solution without pre-defined problem statements or success criteria—and success or failure may not be revealed for quarters or years.

# 4 Career Progression Summary

As noted in the descriptions above, the meaning of each facet changes as you progress through your career. In addition, the relative emphasis of each facet changes across the levels. The following visual approximates this change.




*This figure is meant to be illustrative of the broad changes over the course of the career. Do not use this figure to derive prescriptive ratios among the categories, as the proportions will vary for each individual depending on circumstances.*

At lower levels, you achieve impact by balancing your immediate impact through Code, Operations, and Architecture against the need to Learn—in effect, at lower levels the company is choosing to invest in you now in order for you to deliver more value later. At mid levels, you spend most of your time exercising Code, Operations, and Architecture to deliver value—while you never stop learning, this necessarily becomes a secondary focus. Furthermore, your emphasis among the Code/Operations/Architecture trifecta will shift increasingly from Code and Operations to Architecture as you grow. At higher levels, the responsibilities of Leadership occupy more of your time. Eventually, you exercise your accumulated experience to shape and define Strategy.

The inflection that happens at P5 is significant. You must learn how to make resounding impact above and beyond the individual team level. While your foundation remains your deep technical expertise, you must be intentional to magnify the impact of that expertise: when you write code, it's to blaze new trails or solve exceptional problems; when you design, you are building foundational architectures that cut across existing teams, systems, or domains; you spend much of your time influencing and shaping the work of others. Strategy quickly becomes a primary focus.

Note that Communication is the one facet that doesn't appear in this diagram: effective communication underpins everything you do. You don't trade time in one dimension for more time communicating; how

effective you are in a dimension is determined (in part) by how well you listen to others and express your ideas.

# 5 Level Descriptions

The following level descriptions define characteristics that embody each stage of the Software Engineering ladder. Each is intended to be read and digested independently, so that success is defined and understood at each level. Comprehensively, it provides an understanding of role evolution as well.

The relative importance of each facet changes across the levels. The facets are ***intentionally ordered*** within each level from greatest emphasis to least emphasis.

---

## 5.1 P1: Software Development Engineer Apprentice

*Key Facets: Learning, Code*

You are new to software development, with limited formal education in software. While you contribute by delivering small features (or parts of features), your primary concern is to grow by developing your coding craft. When you deliver, you will be working hand-in-hand with a mentor(s) to author code while learning foundational concepts and practices.

### Learning

You are learning the fundamentals of authoring code—and you are eager and able to do so, absorbing information and internalizing guidance. You are learning how to leverage AI-driven tooling to understand new code bases, debug problems, and avoid rote work. You are curious. Your mindset is that of someone keen to grow and to prove that growth. You take care to learn from feedback and incorporate it into future work.

### Code

You are new to coding, with limited formal experience. You are becoming fluent in a programming language, and you are learning core concepts such as data structures, algorithms, and design patterns. You are learning the software development cycle, including authoring tests to verify correctness and debugging failures.

Fluency is key to moving to the next level, and as you progress as a P1 you are able to demonstrate that increased fluency. Your code style, organization, and documentation are improving over time, and the quality and thoughtfulness of your tests is earning more and more trust that you deliver what you intend to deliver. You are becoming more familiar with subsets of your team's codebase, increasing your productivity. You work independently more and more often.

### Operations

You learn how the team uses metrics and other observability mechanisms to monitor system health and business results. You seek to understand how your changes may impact production behavior and how this behavior is best observed. You can speak to how the team triages issues and manages incidents.

### Communication

You are learning how to communicate technical information. You are effective at communicating knowledge gaps to your mentors—and you're good at asking questions.

### Architecture, Leadership, Strategy

You are developing an awareness of these concepts but this is not a focus.

---

## 5.2 P2: Software Development Engineer-I

*Key Facets: Code, Learning, Operations*

You are familiar with authoring software, but you are new to doing so in a professional context. You contribute by executing specified units of work with consistent and timely quality. You rely on the team and your manager to provide guidance around what work you undertake, the specifications of the design, and the quality of the deliverables. You deliver on your commitments and are transparent about challenges and delays, asking for help to unblock yourself when necessary. You grow by refining your craft and learning the ropes of the industry.

### Code

You are a regular and effective contributor to your team, solving clearly-scoped problems with code. You leverage AI coding assistants to iterate quickly when learning new code bases, investigating problems, and authoring or altering code; you assess generated suggestions against the expected quality bar regarding maintainability, performance, and best practices. You iterate on a solution with more senior team members, incorporating their feedback through conversations and code reviews.

You leverage data structures, algorithms, design patterns, and test methodologies. You identify trade-offs among approaches. You uphold the standards set by your team, routinely delivering code that is tested, easy to read, performant, and documented.

### Learning

You are an expert learner. You recognize that you are new to a profession, and you actively seek out and welcome feedback in all aspects. You leverage AI to explore options and reason about trade-offs—you seek to learn from, rather than simply use, what is suggested. You proactively seek to validate the choices you have made and reinforce what you have learned by seeking feedback from seniors.

You are learning how to apply theories of code organization, maintainability, and adaptability in practice. You are observing and asking questions about your team's architecture, seeking to understand trade-offs and limitations. You are becoming practiced in the varied behaviors and responsibilities expected of an SDE: how to operate software, interact with Product, report status, escalate for help, follow the planning rhythms of your team, estimate work, and share constructive feedback.

### Operations

You are an effective member of your oncall rotation, able to triage and solve simple issues. You understand and

execute your team's runbooks, and you are trusted to escalate when encountering incidents that fall off script. With each rotation, you become more familiar with the behaviors of the software you operate and become more effective at resolving issues. You understand how to reason about metrics, and work with the team to define and modify metrics as your systems change. You are a steward of the team's onboarding experience, seeking to ease the process for future team members.

#### Architecture

Your problem solving tends to happen in the realm of code rather than architecture. However, you are observing and asking questions about architecture, developing a fluency in terms, concepts, and models.

#### Communication

You are an active participant in team conversations, asking questions and making suggestions. You clearly communicate technical information—how code behaves, the implications of your changes, the behaviors of the systems you operate, etc.—to other members of your team. You author documentation for your outputs such as code comments, commit descriptions, and ticket updates. You are learning to deliver meaningful feedback in code reviews.

#### Leadership

Leadership at this level is primarily self-responsibility: you seek to earn the trust of your teammates and manager by taking a proactive role in your personal development. You balance independent delivery—solving problems yourself—with seeking guidance from seniors. When you don't understand best practices or struggle to identify a solution, you seek clarity by reaching out to others. You contribute to estimating the time it will take to complete your tasks. You find opportunities to more actively participate in team processes in order to better understand them. You hold yourself accountable to team decisions.

#### Strategy

You understand how your work impacts the team. You seek to understand how your team fits into your organization and the larger company. You work with your manager and seniors to find opportunities to observe and participate in strategy discussions.

---

## 5.3 P3: Software Development Engineer-II

*Key Facets: Code, Operations, Architecture*

You are a routine and effective contributor, important to the team's collective success. You excel at independently delivering clearly-scoped units of nontrivial work with timely quality, consistently. You design solutions to problems with well-defined problem statements, and you act to help juniors meet commitments. You are active in team discussions and respected for your opinions, helping inform how the team solves collective problems. You grow by thinking about broader system design and developing early leadership skills.

### Code

You are a routine problem solver, independently authoring high-quality code to solve non-trivial problems in a timely and consistent manner. You effectively use AI coding assistants to expedite your workflows. You can better judge when to trust AI-generated suggestions and when to validate them—and you rely less on senior engineers to act as a backstop as a result.

You identify trade-offs, express these cogently, and document your decisions. You act on approved designs, translating intent into functioning and effective systems. You understand the importance of refactoring existing code for better performance, testability, and readability. Your code leverages patterns and test methodologies to enable iterative and autonomous development. You consider the legacy of the code you create—you don't just solve today's problem but anticipate the needs of future developers. You appropriately assess the complexity of your work and meet your commitments.

### Operations

You are a trusted member of your oncall rotation, resolving non-trivial issues with minimal guidance. You help juniors resolve issues they don't understand, and you are trusted to escalate to seniors when you need help. You proactively shape team metrics, alarms, runbooks, and dashboards to improve the operational stance of the team. You consider observability as a first-order concern in the changes you make, ensuring that system performance and health are effectively measured.

### Architecture

You independently solve well-scoped technical problems with few 'unknown unknowns'—and you are trusted to identify ambiguities and seek guidance from seniors. You are learning how to design software with clean contracts, strong abstractions, and clear roles and responsibilities. You identify and articulate trade-offs among competing approaches, and you document your decisions. You contribute to team architectures by engaging in design discussions, proposing alterations, and considering cross-system interactions when authoring code.

### Learning

Learning remains central to this level: you are actively trying to learn the skills necessary to grow to the next level. You seek feedback and incorporate it regularly. You remain intent on becoming an expert in the crafts of coding and operations, learning how to better leverage specific frameworks and technologies, employ AI-assisted tooling, organize complex code, debug difficult problems, and apply best practices around CI/CD and metricing. Furthermore, your curiosity extends beyond the immediate problems and patterns of work that define your day-to-day: you are impatient with learning only 'by doing', and you proactively seek out new ideas, patterns, tools, and approaches.

### Communication

You clearly communicate technical information to others—you train teammates, explain particulars to stakeholders, and communicate progress. You author design documents that capture trade-offs and justify choices. When you review someone's work, you seek to teach through constructive conversation rather than contradiction. You listen to juniors, taking time to gather their input and elevate their voice.

### Leadership

You are an active contributor to team processes and artifacts. While you excel at independent delivery, you understand that your contributions are but one element in a larger team context. You are an active owner of that context, helping maintain and improve the efficacy of your team. The trust you have earned is rooted in your ability to communicate your approach, assumptions, trade-offs, and solution; you escalate early, never hiding problems. You digest problems and propose solutions to seniors, rooting your designs in a rigorous analysis of trade-offs, edge cases, and risks. Management and seniors trust you to accurately represent specifics, and your judgment is relied upon when you hold particular context. You are an active and effective participant in team meetings, agile processes, and planning rhythms.

### Strategy

You proactively contribute to team conversations around solutions and prioritization, seeking to both understand and influence how your team solves problems on behalf of the organization. You identify opportunities for improvement (better experience, efficiencies, performance) relevant to your scope (particular feature, component, or subsystem) and surface them to the team.

---

## 5.4 P4: Senior Software Development Engineer

*Key Facets: Code, Architecture, Operations*

You are a key contributor to your team, leading the way on the most important initiatives, designs, and operational events. You coordinate closely with others to deliver value across multiple services or key components. You habitually author effective architectures, deliver critical bodies of code, and shepherd the quality of the team's outputs. You are an expert on your team's systems, processes, and business domain—and you understand how these fit into the larger cross-team architecture. You contribute to projects that cut across teams and domains. You are relied upon to resolve critical operational incidents, meet essential deadlines, and propose effective designs.

### Code

As a senior engineer, you are an expert practitioner of your coding craft, setting high standards for your team's codebase. You are trusted to take the lead on the most complex and challenging work. Patterns, principles, and best practices are second nature—you are a steward of the team's codebase, and you cultivate it to maintain quality. You understand that less can be more, valuing simplicity in approach. You actively seek out opportunities to refactor and optimize code for better scalability, performance, and readability. You ensure that your team's work is rigorously tested using repeatable, automated patterns. You are skilled at integrating new frameworks, libraries, technologies, and components.

You leverage AI coding assistants to amplify the value of your hard-earned expertise and deep domain knowledge. These tools are a natural extension of your workflow: you know when to trust AI-generated code and when to rely on your own judgment to revise, adapt, and dig deeper. You seamlessly correct output, pivot direction, or refine solutions as needed, maintaining clarity and code quality throughout. You also recognize when AI tools are hindering more than helping and set them aside in favor of your own expertise.

### Architecture

You are an expert in multiple subdomains owned by your team, commanding a deep understanding of key architectural paths. You influence and shape team architecture, working closely with Principal Engineers and other Seniors to design, adapt, and grow your architecture to respond to changing business needs. You treat testability, resiliency, and observability as first-order architectural concerns. You are deeply familiar with industry patterns and technologies relevant to your domain, and you seek to import these as is useful.

### Operations

You are an expert at operating your system. When unforeseen operational events happen, you are relied upon to diagnose and resolve them—and ensure they don't happen again. You know the idiosyncrasies, weaknesses, and limitations of your software. Proactively, you set the standard for dashboards, metrics, alarms, and runbooks; reactively, you resolve the most difficult and critical issues. You identify patterns and trends in order to inform future investments in code and architecture. You share your hard-earned knowledge through mechanisms that scale out beyond yourself.

### Leadership

You are respected for your expertise and are sought out by others for feedback and advice, both internal and external to your team. You shape team practices and processes, proactively proposing changes, exploring new options, and seeking to learn from others (both in and outside the company). You act as a focal point for projects: coordinating deliverables, writing designs, ensuring accurate estimates, reviewing artifacts, and summarizing progress to your manager. You embrace the responsibility of ensuring everything keeps working day to day. This may mean unblocking a merge request or diving into a ticket; it may mean escalating on technical debt or authoring a design proposal. You mentor juniors, improving their understanding of the craft and seeking to better integrate them into team culture and process.

### Communication

You communicate consistently and effectively with technical and nontechnical staff, including Product and Design. You cultivate the ideas of your teammates. You seek to persuade instead of dismiss, treating alternative proposals as opportunities to learn and refine. You listen to their concerns, feedback, and suggestions with an open mind. You document your decisions clearly, articulating trade-offs and rationale in order to inform future stakeholders—you are careful to preserve context in order to conserve hard-won wisdom.

### Strategy

You are deeply familiar with how your team contributes to the organization's (and the company’s) overall strategy, and you actively seek to shape your team’s deliverables to best meet those needs. Grounded in a clear understanding of the customers you serve, you connect technical decisions to meaningful experience improvements and broader business impact. You work with Principal Engineers on problems of indeterminate and variable scope, providing context and identifying gaps. You decompose projects into independent work streams for juniors and are accountable for both delivery and quality.

### Learning

This is a career role, meaning that there is no expectation to move to the next level. If you relish technical delivery

and seek to become an expert in the crafts of code, team-scale architecture, and operational excellence, that's great—this is for you! Therefore, at this level learning takes on a new meaning: you shift from striving to become competent in your craft to seeking true excellence. The technology landscape is deep and always in flux—you are forever curious, for each day brings only more to discover, explore, and master.

---

## 5.5 P5: Principal Engineer

*Key Facets: Leadership, Architecture, Strategy*

You deliver critical business value by solving complex problems that span teams and organizations. You partner with fellow leaders to identify problems and align on their nature; you are an expert in your craft and solve these problems with elegant solutions. You are respected as a technical leader of your domain, demonstrating deep expertise and embodying engineering excellence.

Impact is paramount, and you must scale. Sometimes, your impact will be greatest when you go deep to author critical code or uncover a subtle flaw; at other times, your impact will be greatest when you act broadly to shape approaches across several teams or systems. You are trusted to exercise your judgment—you take the initiative, and you are skilled at communicating why, what, and how to your leadership.

### Leadership

You are beginning to step out of a delivery-first mindset into a leadership-first mindset. To others, you are the engineering face for your team(s): Principal Engineers, Product Managers, and others all seek you out and trust you to represent the technical capabilities and constraints of your space. You guide your teams through change, whether technological, architectural, organizational, or ways of working. You set a tenor of excellence by upholding a high-quality bar without alienating others, seeking to cultivate and mentor.

You seek excellence in all things, such that your systems are archetypes of flexibility, maintainability, security, and testability. You understand that your impact scales with your influence: you cannot uphold these standards solo, but instead must rely on the team to embody them even when you're not looking. You actively promote and model collaboration, listening to feedback from others and successfully mediating contentious technical discussions. When you identify workstreams in large projects, you guide juniors to effective solutions and inspect their estimates.

### Architecture

You are the expert at your domain, and you leverage this expertise to build an architecture that is flexible, performant, and efficient to the needs of the business. You seek to build and cultivate *capabilities* over *features*: you understand that a curated set of fundamental strengths enable a great variety of particular features, quickly and affordably. You build systems that anticipate change; you enable an architecture where it is easy to disband and replace particular systems when the need arises. You encapsulate complexity from dependent teams, isolating the particulars of your architecture from their implementations. You consider testability across systems as a first-order concern, and you build systems that are easy for others to integrate with and test against.

### Strategy

You solve problems of indeterminate and variable scope. The feedback cycles are longer than before, stretching across quarters and into years—a solution must not only deliver incremental value, but withstand the test of time via resiliency, scale, and adaptability. This longer iteration cycle has a consequence: the cost of being wrong is higher. You offset this by leveraging your experience to effect high-judgment decisions, deliberately seeking signals early and identifying reversible decisions.

You are expected to identify, define, socialize, and break down novel problems in your organizational domain. You know that your domain is interconnected with the broader business, and you seek to root your solutions in that larger context. You understand when and how changes to your solutions will better enable business strategy—not just for your immediate domain but interdependent ones as well. You develop partnerships with key stakeholders, seeking to influence designs across team boundaries and proactively inform Product roadmaps. You participate in the planning rhythms of your organization. You are an expert in your business domain, shaping technical investments to unlock long-term business capabilities.

### Code

While architecture is becoming your primary focus, you remain a key contributor on critical code paths. You are not expected to author better code than a Senior Engineer. Rather, you are expected to exercise your judgment to deliver maximum value when you contribute code. Your code is deliberate, innovative, and aligned with broader strategic objectives. You are adept at using AI tools to blaze new trails quickly, testing out novel approaches or validating ambitious ideas to supercharge the business. You set and uphold high standards for organization, patterns, and best practices across your organization.

### Communication

You demonstrate effective verbal and written skills, communicating with everyone from the junior-most engineer to the senior leaders of your organization. You cultivate collaboration, listening to and integrating feedback across all levels. Your design documents are models of reasoned decision making, persuading the reader to accept a conclusion through the careful specification of the problem, thorough presentation of context, and rigorous analysis of trade-offs. You use data to bring clarity to contentious issues.

### Operations

You are an operational leader, seeking to maximize your impact. At times, this may mean applying your expertise to resolve issues that challenge seniors or cut across teams. At other times, it may mean upholding standards through inspection, working with management to tweak processes, or blazing a path by building novel tools or dashboards. You share lessons with the larger engineering community. Furthermore, you treat operations as an architectural concern, treating operational health as a first-order design requirement.

### Learning

While you are an expert in your craft, you continue to learn and grow. You relentlessly seek to better understand the particulars of your dependencies, your customers, and how to influence others across your organization/the company. You are comfortable stepping into a new domain (code, product, strategy) and ramping up quickly. You look outside Zillow—you know most problems are not new. You learn from the larger industry to improve and

innovate by examining case studies, exploring new technologies, and importing best practices. You proactively stay abreast of the latest tooling, test new methodologies, and adapt to new patterns of work.

---

## 5.6 P6: Senior Principal Engineer

*Key Facets: Leadership and Strategy*

You act as a technical thought leader for a large organization, influencing senior leaders and individual contributors to define technical strategy, shaping organizational/architectural structure, and establishing the bar for engineering culture. You build, but the scale at which you build has changed: you trade time spent coding to solve higher order concerns, leveraging your experience and judgment to tie together disparate systems into coherent technical strategy. You understand that organization and software architecture are intertwined, and that the act of shaping software at this scale is as much an act of leadership as it is one of authorship.

You are an expert at choosing how best to deliver value to the company—you are judicious and precise with where you dive deep, and you are effective when you do. Your talent for communication is more critical than ever: you must influence indirectly, shaping software that you may rarely (or never) directly touch or projects over which you have no immediate control. The partnerships you build with leadership and your peers are paramount to your success. Your impact is measured by the critical outcomes you achieve through influence, mentorship, and the growth of others.

### Leadership

You are trusted to act as a technical leader for an organization, typically advising at the Senior Director or VP level. Senior Principals, peers, product managers, and others seek you out and trust you to represent the technical capabilities and constraints of your organization. You develop partnerships with stakeholders at multiple levels, from juniors to directors of Product, maintaining rich channels of communication in order to keep a pulse on and influence technical outcomes. You routinely cross organizational boundaries, partnering with peers and Distinguished Engineers to evolve cross-organization contracts and interactions.

You are also responsible for the engineering tenor and culture of your organization. You keep a close pulse on quality of code, design, testing, and operations across your teams, stepping in to elevate and cultivate as necessary. You seek to understand and measure technical debt, bringing attention to management and offering creative options to pay down. You embrace that leadership of others is most effective when it happens in collaboration, and you are not afraid to roll up your sleeves to help out. Leading through change is equally important: you push the boundaries of what's possible through exploration, proof of concepts, and prototypes. Juniors routinely come to you for advice and feedback, and you are skilled at offering it in a way that motivates instead of discourages.

### Strategy

You are a key contributor to your organization's technical strategy. You are recognized as an expert in your business and industry domain, acting as a bridge between the strategic needs of the business and the capabilities, limitations, and investment opportunities of your systems. You work hand-in-hand with leadership to establish multi-year roadmaps, decomposing problems across teams and time scales. You understand that organization and architecture

are two sides of the same coin, and work with management to establish team structures that align with desired architectures.

The problems you solve are usually undefined: as the steward of a large technical domain, you proactively seek to identify ambiguities, gaps, or inefficiencies unnoticed by others. You are an expert at drilling down to the finest technical levels, intelligently leveraging the expertise of juniors but seeking a deep understanding of particulars—when necessary and appropriate. You focus on the emergent behaviors of teams: you understand how changes in one domain affect the others, and you proactively seek to arrive at an ever-more-coherent common architecture. You are skilled at discerning critical detail and ignoring the chaff.

#### Communication

Your effectiveness is underpinned by your exceptional verbal and written communication skills. Your artifacts set the bar for cogent technical communications. You tailor communications to the audience, from non-technical to junior technical to peers. You cultivate collaboration, listening to and integrating feedback across all levels. You are trusted to resolve contentious cross-organizational technical conflicts.

#### Code, Operations, and Architecture

You create elegant architectures that solve complex engineering and product challenges that span organizations. You understand that the shape of the organization and architecture are fundamentally intertwined. You scour organizational models for misalignment, seeking to unify, simplify, and standardize. You seek coherence between your domain(s) and the larger company, leveraging your broad context to align with cross-cutting strategy. You promote reuse and intentional integration while preserving the autonomy necessary to build domain-centric solutions with nimbleness and alacrity. You identify operational pain resulting from misalignment or poor architecture and act to resolve it.

You code and operate less. You do, however, maintain an awareness of your code bases to ensure you understand how your software works and identify potential emergent problems. You're adept at using AI to quickly ramp up in unfamiliar contexts, discover what's possible in practice, and demonstrate new approaches to others through functional examples. You are intentional: when you code you blaze a fresh technical trail or leverage your deep expertise to resolve a critical issue.

#### Learning

Finally, you can learn technical details without the luxury of working closely on an individual code base—though you're still comfortable diving into one and learning the deepest details, if necessary. You are skilled at knowing what you need to learn to drive the technical direction of your organization. You stay abreast of industry trends and technologies, selectively identifying those that offer your organization a strategic advantage.

---

## 5.7 P7: Distinguished Engineer

*Key Facets: Leadership, Strategy, Communication*

You build at company-scale, working hand-in-hand with senior-most leadership to guide key aspects of the

business's technology. You peer into the future, anticipating disruptive opportunities in the changing technology landscape, balancing risk vs reward. You intentionally steward our engineering culture, cultivating excellence at all levels and across all outputs. You exemplify technical leadership at the largest scales, acting as a paragon for others to aspire to.

### Leadership

You operate at the company scale, typically advising at the Vice President or Senior Vice President level. As a trusted member of the senior leadership team, you work hand-in-hand with executives and Senior Principal Engineers to achieve strategic business objectives. You maintain a pulse on fundamental product ideation in order to inform the capabilities of the company's engineering ecosystem, proactively seeking to remedy deficiencies and strategic bottlenecks. Your domain expertise—both technical and business—are recognized and relied upon across the company.

You set the standard for engineering excellence. You hold management accountable for engineering quality, actively educating leadership as to the costs of deferred investments, accumulating technical debt, and outdated technologies. You seek to educate and disseminate knowledge—not only your own but, more importantly, that of others. You identify mechanisms and promote standards to measure and drive high-quality outputs—but you are protective of the line engineers, seeking to insulate them from well-meaning but intrusive top-down directives. You not only embody the philosophy of cultivation over prescription, and you teach your Principal engineers how to act this way in turn.

### Strategy

You partner with executives to define and act upon key business initiatives, establishing strategy that cuts across organizational boundaries. You survey an expansive technical and business landscape to anticipate strategic needs of the business, identifying and disambiguating the largest, most critical problems. You anticipate the emergent problems that arise from misaligned architectures, charters, and contracts across large organizations, and you leverage your deep technical and domain expertise to achieve elegance and efficiency at company scale.

You act as a critical influence on key technological investments. You participate in acquisition decisions to evaluate potential technology advantages or alignment. You critically assess (and reassess) build-vs-buy decisions, carefully distinguishing differentiated and commodity value. You are patient and empathetic, but you own and steward key technical decisions to their correct outcome regardless of tensions and vested interests. You constantly inspect our current capabilities for sunset opportunities, rigorously identifying areas that are redundant, outdated, or unnecessary.

### Communication

You are exceptional at tailoring your communications: you inform and motivate large bodies of engineers; you crisply inform senior leadership. You are a technical face of the company, both internally and externally, skilled at representing initiatives, capabilities, and technology to a complete gamut of audiences. You cultivate networks of collaboration, building communication channels across the company.

### Code, Operations, and Architecture

You drive cross-organization architecture by identifying key components, defining the relationships between, and delegating specific sub-designs to relevant juniors. You treat organizational structure and architecture as two variants of the same problem, acting as the technical stakeholder in all organizational decisions. You tend to see operational problems through the lens of architecture and process, shaping strategy and culture to solve structural operational pain. Depending on the needs of your organization, you may occasionally lead through code by blazing novel paths.

Finally, you recognize that this is a rare role, and that your actions shape it as much as any role description. You don't just follow the expectations of a Distinguished Engineer; you seek to shape and elevate that role in your daily actions. You are a core leader of our company, and you do not hesitate to lead.

### Learning

You leverage others to learn the information necessary to lead and strategize across hundreds of engineers. You are an expert at knowing how to learn as much as necessary and no more—sometimes, this means going very deep, judiciously. You stay abreast of industry trends and technologies, anticipating risks and opportunities for the company.