**Chapter 3. Creating the Big Picture**

Here’s my favorite story about a group that was missing a big picture. An organization I was in had an upcoming all-hands meeting. People could propose topics in advance, and there had been a question about a critical system—let’s call it SystemX—that had caused some outages. I’d been tagged to respond. While I prepared my talking points, I got three almost simultaneous DMs:

* The first: “Please reassure everyone that we know SystemX has been a problem, but we’re staffing up the team that supports it and adding replicas to help it scale. We don’t anticipate more outages.”
* At the same time: “So glad someone asked that question! We should emphasize that SystemX has been deprecated and everyone should plan to move off it.”
* And: “Hey, could you tell everyone that I’ve set up a working group to explore how to evolve SystemX. We’ll announce plans next quarter. If anyone wants to join the working group, they should contact me.”

The public forum would have been a great opportunity to spread awareness of any of these three very reasonable paths forward. But why were there three different plans?

At the end of [Chapter 2](https://learning.oreilly.com/library/view/the-staff-engineers/9781098118723/ch02.html#three_maps), we finished uncovering the existing treasure map of your organization. If your group already has one of those—a single compelling, well-understood goal and a plan to get there—your big picture is complete. You can jump to [Part II](https://learning.oreilly.com/library/view/the-staff-engineers/9781098118723/part02.html#part2), where I’ll talk about how to execute on big projects. But a lot of the time, staff engineers find that the goal is not clear, or that the plan is disputed. If that’s the situation you’re in, read on.

In this final chapter of [Part I](https://learning.oreilly.com/library/view/the-staff-engineers/9781098118723/part01.html#part1), we’re going to talk about *creating* the big picture. When the path is undefined and confusing, sometimes you need to get the group to agree on a plan and create the missing map. This map often comes in the form of a *technical vision,* describing a future state you want to get to, or a *technical strategy,* outlining how you plan to navigate challenges and achieve specific goals. I’ll open by describing both of these documents, including why you’d want each one, what shapes they might take, and what kinds of things you might include in them. Then we’ll look at how to work as a group to create this kind of document. We’ll look at the three phases of creating the document: the approach, the writing, and the launch.[**1**](https://learning.oreilly.com/library/view/the-staff-engineers/9781098118723/ch03.html#ch01fn28) Finally, we’ll work through a fictional case study to see some of those techniques in action. Let’s lay that scenario out now, so you can start thinking about how *you’d* approach it.

**The Scenario: SockMatcher Needs a Plan**

SockMatcher formed a few years ago as a two-person startup aiming to solve an important problem: odd socks. People who have lost a sock upload an image or video using the company’s mobile app, and a sophisticated machine learning algorithm on the backend attempts to find another user who has lost one of an identical pair. If one of the two sock owners wants to sell their odd sock to the other, the algorithm then suggests a price. Every change in sock ownership is tracked in a distributed sockchain ledger.

As you might imagine, venture capitalists went wild for it.[**2**](https://learning.oreilly.com/library/view/the-staff-engineers/9781098118723/ch03.html#ch01fn29) SockMatcher quickly grew into the largest odd-sock marketplace on the internet. The company has expanded to add partnerships with several bespoke sock manufacturers, personalized sock recommendations, and even gloves and buttons. It launched an external API that third parties can use for sock analysis as a platform (SaaaP). Customers love the new features.

The company’s architecture has grown organically. It’s all built around a single central database and data model, with a monolithic binary managing login, account subscriptions, billing, matching, personalization, image and video uploads, and so on. Product-specific logic is built into each of these functions. For instance, the billing code includes logic for how customers are charged: per successful match for socks, but as a quarterly subscription for buttons. Sock data and customer data are stored in the same large datastore, which includes sensitive personally identifiable information (PII) about customers, like their names, credit card numbers, and shoe sizes.

For competitive reasons, SockMatcher has prioritized getting new features into the apps quickly, rather than building in a scalable or reusable fashion. For example, the team implemented the gloves feature as a special case of the existing sock-matching functionality, adding a field to the sock data model to allow marking an item as “left” or “right.” When a user uploads an image of a glove, the software generates a mirror image, then treats the glove as just another kind of sock.

When the company decided to add button matching, several senior engineers argued that it was time to rearchitect and create a modular system where it would be easy to add new types of matchable objects. Business pressures won out, though, and button matching was also implemented as a special case of socks, with new fields in the data model to allow specifying the number of buttons in a set and the number of holes per button. The billing code, personalization subsystem, and other components contain hardcoded custom logic to handle differences in socks, gloves, and buttons, mostly implemented as *if* statements scattered throughout the codebase.

Now there’s a new proposed business goal: the company wants to expand to match food storage containers and lids. This product will have different characteristics from existing ones. Unlike socks, containers and their lids aren’t identical. The team will need different matching models and logic and a whole new set of vendors and partnerships so that they can offer the customer a brand-new replacement lid or container when no match is available. The company’s most recent product strategy deck speculates about adding earrings, jigsaw-puzzle pieces, hubcaps, and more in the future.

The new food storage container team is ready to start scoping out the feature. They’re not eager to begin working in the existing monolith: they really want to build their own independent matching microservice with their own datastore. But even if they do, they’ll need code for authentication, billing, personalization, safely handling PII, and other shared functionality—all of which are currently optimized for the sock model. If they want to work autonomously, they’ll need to expose this functionality from the monolith or reimplement it. Either will take time, so they anticipate some pressure to declare food storage containers to be a kind of sock and to work within the existing code, adding more edge cases alongside gloves and buttons where needed. The team is split on what the right next step is.

There are some other challenges:

* The API that was shared with third parties isn’t versioned, and so it’s difficult to change it; with new integrations planned, this problem will get more difficult to solve the longer it’s left.
* The homegrown login functionality has always been, to quote the engineer who built it three years ago, “kind of janky.” It’s got a few years of growth left in it, but it’s not code anyone is proud of.
* The matching functionality is the best on the market and makes customers happy, but there are times when it fails to find a match even though one is available.
* One team member has an idea for a new algorithm and system that will find matches in a fraction of the current time. They’re really excited about it.
* The team responsible for operating the monolith hasn’t been able to keep up with its growth and is reacting constantly to scaling problems. They’re paged several times every day for full disks, failed deploys, and software bugs.
* With more and more engineers working in the same codebase and reusing existing functionality, there’s more unexpected behavior, and user-visible bugs are being pushed more often. Almost every team and user is affected by almost every outage.
* Celebrities and influencers selling their socks have caused 100x spikes in user traffic, leading to complete outages. The food storage container launch might attract celebrity chefs to the platform, further increasing demand.
* Every new piece of functionality slows the monolith’s build time, and unowned, flaky tests aren’t helping: it typically takes three hours to build and deploy a new version, lengthening most incidents.
* App Store reviews for the mobile app have begun to trend downward; many of the one-star reviews note that availability has been poor.

Although this scenario includes a lot of problems, many of them have straightforward technical solutions. Every new person who joins the company suggests changes: sharding the datastores, versioning the APIs, extracting functionality from the monolith, and so on. Various working groups have kicked off. They always start as a room of 20 people who care deeply but don’t agree, then get mired in no one having time to focus on them. After all, there’s feature work to do that feels more important or more likely to succeed. The engineering organization can’t seem to get momentum behind any single initiative.

What would you do? We’ll return to this scenario at the end of the chapter.

**What’s a Vision? What’s a Strategy?**

Should you build new functionality as a reusable platform or as part of a specific product? Should teams learn the difficult new framework or stick with the popular deprecated one? Maybe each team can make their own decision: decentralized decision making can let organizations move more quickly and solve their own problems. But there can be disadvantages when each team decides for itself:

* There can be a [“tragedy of the commons”](https://oreil.ly/KEPbi), where teams pursuing their own best action without coordinating with others leads to an outcome that is bad for everyone. There’s that local maximum again.
* Shared concerns can be neglected because no one group has the authority or an incentive to fix them alone.
* Teams may be missing enough context to make the best decision. The people taking action might be different from the people experiencing the outcomes, or might be separated in time from them.

When there are major unmade decisions, projects get slowed or blocked. Nobody wants to delay their project for the sake of a long and painful fight to make a controversial decision or choose a standard. Instead, groups make locally good decisions that solve their own immediate problems. Each team chooses directions based on their own preferences or on rumors about the organization’s technical direction—and often embed those choices in the solution they create. Postponing the big underlying questions makes them even harder to solve down the line.[**3**](https://learning.oreilly.com/library/view/the-staff-engineers/9781098118723/ch03.html#ch01fn30)

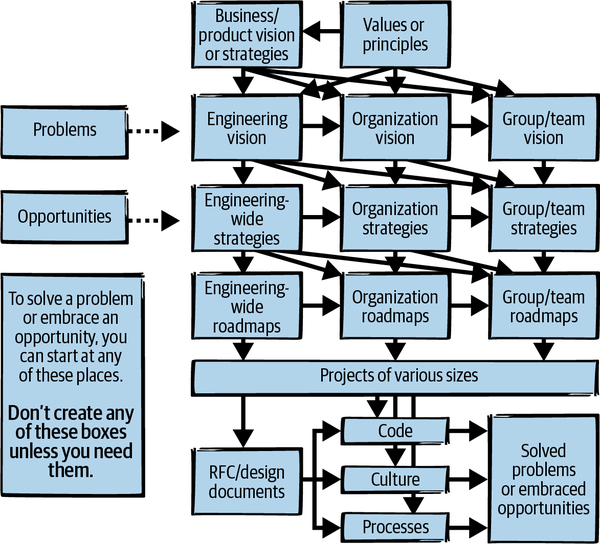
When organizations realize they need to solve some of these big underlying problems, the words *vision* and *strategy* get thrown around a lot. You’ll hear these used both interchangeably and to mean distinct things. To avoid confusion over terminology, let’s start with some working definitions we can use throughout this chapter.

**What’s a Technical Vision?**

A *technical vision* describes the future as you’d like it to be once the objectives have been achieved and the biggest problems are solved. Describing how everything will be *after* the work is done makes it easier for everyone to imagine that world without getting hung up on the details of getting there.

You can write a technical vision at any scope, from a grand picture of the whole engineering organization down to a single team’s work. Your vision may inherit from documents at larger scopes, and it may influence smaller ones (see [Figure 3-1](https://learning.oreilly.com/library/view/the-staff-engineers/9781098118723/ch03.html#depending_on_the_size_of_the_problemcom) for some examples).

A vision creates a shared reality. As a staff engineer who can see the big picture, you can probably imagine a better state for your architecture, code, processes, technology, and teams. The problem is, many of the other senior people around you probably can too, and your ideas might not all line up. Even if you think you all agree, it’s easy to make assumptions or gloss over details, missing big differences of opinion until they cause conflict. The tremendous power of the written word makes it much harder to misunderstand one another.



**Figure 3-1. Depending on the size of the problem, you might start with an engineering-wide vision, a team-scoped vision, or something in between. Don’t create a vision, strategy, etc., unless you need it.**

A technical vision is sometimes called a “north star” or “shining city on the hill.” It doesn’t set out to make *all* of the decisions, but it should remove sources of conflict or ambiguity and empower everyone to choose their own path while being confident that they’ll end up at the right place.

**RESOURCES FOR WRITING A TECHNICAL VISION**

If you’re setting out to write a technology vision document, here are some resources I recommend:

* [*Fundamentals of Software Architecture*](https://oreil.ly/RLl0h) by Mark Richards and Neal Ford (O’Reilly)
* Chapter 4 of [*Making Things Happen*](https://oreil.ly/a9lx1) by Scott Berkun (O’Reilly)
* [“How to Set the Technical Direction for Your Team”](https://oreil.ly/zhD2Q), by James Hood of Amazon
* [“Writing Our 3-Year Technical Vision”](https://oreil.ly/Rew44), by Daniel Micol of Eventbrite

There’s no particular standard for what a vision looks like. It could be a pithy inspirational “vision statement” sentence, a 20-page essay, or a slide deck. It might include:

* A description of high-level values and goals
* A set of principles to use for making decisions
* A summary of decisions that have already been made
* An architectural diagram

It could be very detailed and go into technology choices, or it could stay high-level and leave all of the details to whoever is implementing it.

Whatever you create, it should be clear and opinionated, it should describe a realistic better future, and it should meet your organization’s needs. If you could wave a magic wand and be done, what would your architecture, processes, teams, culture, or capabilities be? That’s your vision.

**What’s a Technical Strategy?**

A *strategy* is a plan of action. It’s how you intend to achieve your goals, navigating past the obstacles you’ll meet along the way. That means understanding where you want to go (this could be the vision we just discussed!) as well as the challenges in your path. When I use the word *strategy* in this chapter, I always mean a specific document, not just being a strategic sort of thinker.

A technology strategy might underpin a business or product strategy. It might be a partner document for a technical vision, or it might tackle a subset of that vision, perhaps for one of the organizations, products, or technology areas it encompasses. Or it might stand entirely alone.

Just like a technical vision, a technical strategy should bring clarity—not about the destination, but about the path there. It should address specific challenges in realistic ways, provide strong direction, and define actions that the group should prioritize along the way. A strategy won’t make all of the decisions, but it should have enough information to overcome whatever difficulties are stopping the group from getting to where it needs to go.

**RESOURCES FOR WRITING A TECHNICAL STRATEGY**

The canonical book on strategy is *Good Strategy/Bad Strategy* by Richard Rumelt (Currency). I recommend taking the time to read it if you can. Other great resources include:

* [“Technical Strategy Power Chords”](https://oreil.ly/EDODP) by Patrick Shields
* [“Getting to Commitment: Tackling Broad Technical Problems in Large Organizations”](https://oreil.ly/RKUwO) by Mattie Toia
* [“A Survey of Engineering Strategies”](https://oreil.ly/tF2TU) by Will Larson
* [*Technology Strategy Patterns*](https://oreil.ly/31lii) by Eben Hewitt (O’Reilly)
* [Rands Leadership Slack](https://oreil.ly/O4bad), specifically the channels #technical-strategy and #books-good-strategy-bad-strategy

Just like a technology vision, a strategy could be a page or two or it could be a 60-page behemoth. It will likely include a diagnosis of the current state of the world, including specific challenges to be overcome, and a clear path forward for addressing those challenges. It might include a prioritized list of projects that should be tackled, perhaps with success criteria for those projects. Depending on its scope, it could include broad, high-level direction or decisions on a specific set of difficult choices, explaining the trade-offs for each one.

In *Good Strategy/Bad Strategy*, Rumelt describes “the kernel of a strategy”: a diagnosis of the problems, a guiding policy, and actions that will bypass the challenges. Let’s look at each one.

**The diagnosis**

“What’s going on here?” The diagnosis of your situation needs to be simpler than the messy reality, perhaps by finding patterns in the noise or using metaphors or mental models to make the problem easy to understand. You’re trying to distill the situation you’re in down to its most essential characteristics so that it’s possible to really comprehend it. This is *difficult*. It will take time.

**Guiding policy**

The guiding policy is your approach to bypassing the obstacles described in the diagnosis. It should give you a clear direction and make the decisions that follow easier. Rumelt says it should be short and clear, “a signpost, marking the direction forward.”

**Coherent actions**

Once you’ve got a diagnosis and guiding policy, you can get specific about the actions you’re going to take—and the ones you won’t. Your actions will almost certainly involve more than technology: you might have organizational changes, new processes or teams, or changes to projects. I really can’t stress this enough: you’ll commit time and people to *these* actions rather than to the long list of other ideas that were on the table at the start. This kind of focus will likely mean that you and others don’t get to do some things you’ve been excited about. It is what it is.

A strategy should draw on your advantages. For example, when Isaac Perez Moncho, an engineering manager in London, was writing an engineering strategy for a company, he looked to create positive feedback loops. That company’s product engineering teams were facing many problems, he told me: lack of tooling, too many incidents, and poor deployments. But he had an advantage: a great DevOps team who could solve those problems, if only they had more time. His guiding policy was freeing up some of the DevOps team’s time. Making space for them let them automate processes and free up even more time, creating a positive feedback loop that made them available to solve other problems. Think about ways to amplify your advantages in a self-reinforcing way.

Finally, a strategy isn’t an aspirational description of what someone else would do in a perfect world. It has to be realistic and acknowledge the constraints of your situation. A strategy that can’t get staffed in your organization is a waste of your time.

**Do You Really Need Vision and Strategy Documents?**

Technical visions and strategies bring clarity, but they can be overkill for a lot of situations. Don’t make unnecessary work for yourself. If it’s easy to describe the state you’re trying to get to or the problem you’re trying to solve, what you actually want might be more like the goals section of a design document, or even the description of a pull request.[**4**](https://learning.oreilly.com/library/view/the-staff-engineers/9781098118723/ch03.html#ch01fn31) If everyone can get aligned without the document, you probably don’t need it.[**5**](https://learning.oreilly.com/library/view/the-staff-engineers/9781098118723/ch03.html#ch01fn32)

If you’re sure you need something, think about what shape it should take. Adapt to what your organization needs and will support. For example, if a lack of direction is slowing you all down, you might want to get a group together to create an abstract high-level vision, then get more concrete about how to implement it. If you’re preparing for company growth, your CTO might ask you to get a group together from across engineering and describe what your architecture and processes will look like in three years. But if your group is repeatedly getting stuck on a particular missing architectural decision, don’t spend too much time on philosophy: make a call on the specific item that’s blocking you.

Writing technical vision or strategy takes time. If you can achieve the same outcome in a more lightweight way, do that instead. Create what your organization needs and no more.

**The Approach**

Creating a vision, a strategy, or any other form of cross-team document is a big project. There will be a ton of preparation, then a ton of iteration and alignment. Bear in mind that getting people to agree isn’t a chore that stands in between you and the real work of solving the problem: the agreement *is* the work. Any insight or bold vision you’re bringing to the project is only going to be worth anything if you can bring people along on the journey with you. Just like you wouldn’t admire time spent on an engineering solution that ignores the laws of physics, advocating for a project that you know you won’t be able to convince your organization to do isn’t a good use of your time.

Although I’ve talked about strategies and visions separately up until now, for the rest of this chapter I’m not going to distinguish between them much. They’re very different things, but both involve getting people together, making decisions, bringing your organization along, and telling a story. You can use the same approach for creating other big-picture documents, like a [technology radar](https://oreil.ly/lPzPZ) or a set of engineering values.

Creating any of these documents is a classic “1 percent inspiration and 99 percent perspiration” endeavor, but if you prepare properly, you increase your chances of launching something that actually gets used. I’ll talk about some of the prep work that can set you up for success, and then I’ll invite you to think through the outputs of that prep work, evaluate whether you can succeed, and decide whether to make the project official.

**Embrace the Boring Ideas**

I don’t know about you, but when I was new to the industry, I thought that very senior engineers were wizards who would spend their days coming up with insightful game-changing solutions to terrifyingly deep technical problems. I imagined it to be something like a *Star Trek: The Next Generation* episode where there’s an impending warp core antimatter containment failure or what have you, and everyone’s out of ideas and freaking out, but then suddenly Geordi La Forge or Wesley Crusher exclaims, “Wait! What if we <extreme technobabble>” and taps eight characters on a touch screen, and the Enterprise is saved with seconds to spare. Phew!

Real life is a bit different. OK, sometimes “What if we <extreme technobabble>” actually is the answer: especially in very small companies, sometimes you really are stuck until an experienced person drops in to describe a solution and save the day. But if there are senior people around, most likely there are already plenty of good ideas. The gap is getting everyone to agree on what to do.

As you go into this project to create a vision or a strategy, be prepared for your work to involve existing ideas, not brand-new ones. As Camille Fournier, author of *The Manager’s Path*, [wrote](https://oreil.ly/WgEPL):

*I kind of think writing about engineering strategy is hard because good strategy is pretty boring, and it’s kind of boring to write about. Also I think when people hear “strategy” they think “innovation.” If you write something interesting, it’s probably wrong. This seems directionally correct, certainly most strategy is synthesis and that point isn’t often made!*

Will Larson [adds](https://oreil.ly/3TH5a), “If you can’t resist the urge to include your most brilliant ideas in the process, then you can include them in your prework. Write all of your best ideas in a giant document, delete it, and never mention any of them again. Now…your head is cleared for the work ahead.”

Creating something that feels “obvious” can feel anticlimactic when you’re writing it: we’d all love to show up with a genius visionary idea and save the USS *Enterprise*! But usually what’s needed is someone who’s willing to weigh up all of the possible solutions, make the case for what to do and not do, align everyone, and be brave enough to make the (potentially wrong!) decision.

**Join an Expedition in Progress**

If someone is already working on the kind of document you want to create, don’t compete—join their journey. Here are three ways to do that:

*Share the lead*

You can bring leadership to an existing project without taking over. Suggest a formal split that gives each of you a chance to lead in a compelling way. One takes the overall project, for example, while the other leads some individual initiatives inside that work. You could also suggest co-leading: take it in turns to be the primary author, and split up the work as it comes along. If the leaders are enthusiastic about each other’s ideas and are all pushing in the same direction, this can make for a very effective team.

*Follow their lead*

Put your ego aside and follow their plan. If they’re less experienced than you, you can have a huge impact by nudging them in the right direction and helping make the work as good as it can be. Being the grizzled, experienced best supporting actor is an amazing role.[**6**](https://learning.oreilly.com/library/view/the-staff-engineers/9781098118723/ch03.html#ch01fn33) You can use your deep technical knowledge to fill gaps in their skills, for example, spelunking in a legacy codebase to understand exactly how something works. You can also advocate for the plan in rooms they aren’t in. Back them up and help make the thing happen.

*Step away*

A third approach, of course, is to decide the work is going to succeed without you, be enthusiastic about it, and go find something else to do. If the project doesn’t need you, find one that does.

It can be difficult to let other people lead when their direction is not where you’d planned to go. Tech companies’ promotion systems can incentivize engineers to feel like they need to “win” a technical direction or be the face of a project. This competition can lead to “ape games”: dominance plays and politicking, where each person tries to establish themselves as the leader in the space, considering other people’s ideas to be threatening. It’s toxic to collaboration and makes success much harder to achieve.

My friend Robert Konigsberg, a staff engineer and tech lead at Google, always says, “Don’t forget that just because something came from *your* brain doesn’t make it the best idea.” If you tend to equate being right with “winning,” step back and focus on the actual goal. Practice perspective: is their direction wrong, or is it just *different*? Would you advocate just as hard for the path you want if it had been a colleague’s idea? Even if it’s better, be wary of fighting for a marginally better path at the cost of not making a decision at all. As [Will Larson writes](https://oreil.ly/TmbEr), “Give your support quickly to other leaders who are working to make improvements. Even if you disagree with their initial approach, someone trustworthy leading a project will almost always get to a good outcome.”

What if you *don’t* think the person’s ideas or leadership can work, even with your support? While you sometimes need to be flexible, that shouldn’t extend to endorsing ideas you think are dangerous or harmful. Even then, try to join the existing journey and change its direction, rather than setting up a competing initiative from scratch: you’ll have allies in place and momentum already built, and you’ll learn from whatever they’ve done so far.

If there really is no way for multiple people to succeed on the same project without playing ape games, consider going somewhere with more available scope (and a healthier culture).

**Get a Sponsor**

Except in the most grassroots of cultures, any big effort that doesn’t have high-level support is unlikely to succeed. A vision or strategy can begin without sponsorship, but turning it into reality later will be a challenge. Even early on, a sponsor helps clarify and justify the work. If your director or VP is on board with your plans from the start, then what you’re creating is implicitly the *organization’s* treasure map, not just yours—reducing the risk that you’re wasting your time. Sponsorship can also add hierarchy to groups that would otherwise get stuck attempting consensus. The sponsor can set success criteria and act as a tiebreaker when decisions are stuck in committee. They can nominate a lead or “decider”—sometimes called a directly responsible individual, or DRI—who will get the final say when the group is stuck. You don’t necessarily need that, but keep it in mind as an option that’s available to you.

Getting a sponsor might not be easy. Executives are busy and you’re implicitly asking them to commit resources, people, and time to your proposal over something else they’d intended to do. Maximize your chances by bringing the potential sponsor something they want. While a proposal that’s good for the company is a great start, you’ll get further with one that matches the director’s own goals or solves a problem they care about. Find out what’s blocking their goals, and see if the problem you’re trying to solve lines up with those. The sponsor will also have some “objectives that are always true”: if you can make their teams more productive or (genuinely) happier, that can be a compelling reason for them to support your work.

Think about and practice your “elevator pitch” before trying to convince a sponsor to get on board with your project: if you can’t convince them in 50 words, you may not be able to convince them at all. I once tried to talk Melissa Binde, at the time a director of engineering at Google, into sponsoring a project I cared deeply about. I went into all sorts of detail as I tried to make it important to her, too. My spiel wasn’t convincing *at all—*but Melissa kindly took the opportunity to coach: “The way you’re telling this story doesn’t make me care, and it won’t make anyone else care either. Try again—tell the story from a different angle.” She let me try a few different project rationales, and told me which ones resonated. You will almost never get an opportunity like that, so go in with your pitch already polished.

Can a staff+ engineer be a project sponsor? In my experience, no, not directly. A sponsor needs the power to decide what an organization spends time and staffing on, and such decisions are usually up to the local director or VP.

Once you have the sponsorship, check in enough to make sure you *still* have it. Sean Rees, principal engineer at Reddit, says that one of the biggest mistakes a staff+ engineer can make is not maintaining their executive sponsorship: “I think this one is pernicious because you can start sponsored and have that wane as realities change…and then have to navigate the tricky waters of getting back into alignment.”

**Choose Your Core Group**

Some people are very disciplined about setting out on a journey and not getting distracted by side quests, but most of us benefit from a little accountability. Working with other people will give you that accountability. When one person is distracted or flagging, the others will keep the momentum going, and knowing that you’ll need to check in about the work can be a powerful motivator. Working in a group also lets you pool your contacts, social capital, and credibility, so you can reach a broader swathe of the organization. You’ll have to work to get everyone aligned, but since ultimately you’ll need to align with your whole organization, getting your core group to agree can help you uncover potential disagreements early on.

Aim to recruit a *small* core group to help you create the document, as well as a broader group of general allies and supporters. Of course, if you’ve joined someone else’s journey, you’re going to be part of *their* core group instead. (For the sake of simplicity, the rest of this chapter assumes you’re leading the effort.)

Who do you want in your group? Pull out your topographic map. Who do you need on your side? Who’s going to be opposed? If there are dissenters who will hate any decision made without them, you have two options: make very sure that you have enough support to counter their naysaying, or bring them along from the start. Bringing them along will be easier if you understand *why* they’re against the work and what they’d like to see happen instead.

While you may have many colleagues who care about what you’re writing and want to help, keep the core group manageable: two to four people (including you) is ideal. A time commitment can help you here: if everyone who’s part of the core team needs to commit 8 or 12 hours a week to this work, you’ll be able to keep the group small without excluding anybody. (This is a good way to keep “tourists” out of the way, too.) Outside the core team, you can offer more lightweight involvement: you’ll interview them, try to represent their point of view in your work, and let them review early drafts.

Once you have your core group, be prepared to let them work! Be clear from the start about whether you consider yourself the lead and ultimate decider of this project or more of a [first among equals](https://oreil.ly/BKNe3). If you’re later going to want to use tiebreaker powers or pull rank, highlight your role as lead from the start, perhaps by adding a “roles” section to whatever kickoff documentation you’re creating. But whether you’re the lead or not, let your group have ideas, drive the project forward, and talk about it without redirecting all questions to you. Offer opportunities to lead, and make sure you’re supportive when they take initiative. Their momentum will help you move along faster, so don’t hold them back.

As for your broader group of allies, keep them engaged: interview them and use the information they give you. Send them updates on how the document is going. Invite them to comment on early drafts. Your path will be much easier if you have a group of influential people who support what you’re doing, and the knowledge they bring from across the organization will yield better results, too.

**Set Scope**

As you think about the specific problem or problems you’re trying to solve, consider how much they sprawl across the organization. Do you want to influence all of engineering, a team, a set of systems? Your plan’s scope may match the scope of your role as a staff engineer, but might also extend well beyond it.[**7**](https://learning.oreilly.com/library/view/the-staff-engineers/9781098118723/ch03.html#ch01fn34)

Aim to cover enough ground to actually solve your problem, but be conscious of your skill level and the scope of your influence. If you’re trying to make a major change to, say, your networks, you’d be wise to include someone from the networks team in the core group and to build credibility with that team. Otherwise, you’re setting yourself up for conflict and failure. If you’re trying to write a plan for areas of the company that are well outside your sphere of influence, make sure you have a sponsor who has influence there—and ideally some other core group members who have clear maps of those parts of the organization.

Be practical about what’s possible. If your vision of the future involves something entirely out of your control, like a change of CEO or an adjustment in your customers’ buying patterns, you’ve crossed into magical thinking. Work around your fixed constraints, rather than ignoring them or wishing they were different.

That said, if you’re writing a vision or strategy for just your part of the company, understand that a higher-level plan may disrupt yours. Even if you’re writing something engineering-wide, a change in business direction can invalidate all of your decisions. Be prepared to revisit your vision at intervals and make sure it’s still the right fit for your organization. As you make progress on your vision or strategy, you may find that your scope changes. That’s OK! Just be clear that it has.

Be clear, too, about what kind of document you intend to create. I recommend starting by having each of the core group members be really explicit about what documents, presentations, or bumper stickers they hope will exist at the end of the work. Then choose a document type and format that makes sense to you and, most importantly, to your sponsor. If they are enthusiastic about a particular approach, soul-search before doing something else. Don’t make your life harder than it has to be.

**Make Sure It’s Achievable**

As you think through the project ahead, how many big problems do you see? Are there decisions that you really don’t know how to make, or massive technical difficulties? Having one or two problems you don’t know how to solve doesn’t mean you shouldn’t wade in, but have a think about whether the problem is solvable at all.

A practical step you can take here is to talk with someone who’s done something similar before. Ask something like: “I’m writing a vision/strategy and I currently see three problems ahead that I don’t yet know how to tackle. I’m willing to try, but I don’t want to waste my time if this isn’t solvable. Can you give me a gut check on whether I’ll hit a dead end?” Or perhaps: “Everything ahead seems doable and I only have one problem, but it’s that my boss thinks this is a waste of time and wants me to focus on something else entirely. Is this worth continuing?”[**8**](https://learning.oreilly.com/library/view/the-staff-engineers/9781098118723/ch03.html#ch01fn35)

Maybe you think the problem is important and could be solved, but not currently by *you*. Is there a coach or mentor who could help you stretch to do it? Or is this just too big for where you are in your career? If you’re a staff engineer balking at a problem scoped for a principal engineer, that doesn’t reflect badly on you: you’re actually doing pretty good risk analysis.

If, at the end of this analysis, you decide that the problem isn’t solvable, or at least not by you, you have five options:

* Lie to yourself, cross your fingers, and do it anyway.
* Recruit someone who has skills that you’re missing, and either work with them or ask them to lead the project (and give you a subsection of it to hone your skills on).
* Reduce your scope, add in the fixed constraints, and start this chapter again with a differently shaped problem to solve.
* Accept that nobody’s going to write a vision/strategy to solve the problems you can see, conclude that your company will probably be OK without one, and go work on something else.
* Accept that nobody’s going to write a vision/strategy to solve the problems you can see, conclude that your company *won’t* be OK without one, and update your resume.

**Make It Official**

Before we move on, let’s recap those questions we’ve asked along the way. Here’s a checklist to consider before starting to create a technical vision or strategy.

* We need this.
* I know the solution will be boring and obvious.
* There isn’t an existing effort (or I’ve joined it).
* There’s organizational support.
* We agree on what we’re creating.
* The problem is solvable (by me).
* I’m not lying to myself on any of the above.

Introspect a bit on that last question. If you can’t check all of these boxes, my opinion is that you shouldn’t continue. There’s a high opportunity cost if you spend your time on a vision or strategy instead of any of the other work that needs a staff+ engineer.

If you do feel ready to go, though, here’s one final question: are you ready to commit to the work and start working on it “out loud”? This might be a good time to formally set up the vision or strategy creation as a project, with kickoff documentation, milestones, timelines, and expectations for reporting progress. If you have *any* tendency to procrastinate or get distracted, these structures are especially important.

Your level of transparency here will depend on your knowledge of your organization: think about the topographical map you made last chapter. If you can be open about work like this, it will be easier for people to bring you information and gravitate toward you to help. If you feel you need to create a vision or strategy in secret, understand why. Does it mean you don’t have enough support? Are you unsure of your own level of confidence and commitment? If you need to do a bit of the work first to convince yourself that you’re going to stick with it, well, I won’t judge, but make it official as soon as you can. If you set everyone’s expectations, you’re less likely to meet with competing efforts—or at least you’ll find out about them early.

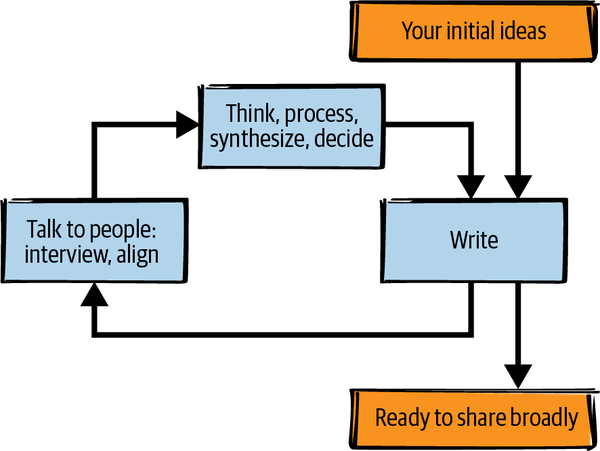
**The Writing**

The prep work is done, the project is official, you’ve got a sponsor, you’ve got a core group, and you’ve chosen a document format. The work you’re doing is framed and scoped. Time to start writing the document for real.

**The Writing Loop**

In this section, I’m going to talk through some techniques for actually creating your vision, strategy, or other broad document. We’ll look at writing, interviewing people, thinking, and making decisions, as well as staying aligned while you do it. These techniques won’t necessarily happen in the order I’m listing them. In fact, probably you’ll do most of them many times (see [Figure 3-2](https://learning.oreilly.com/library/view/the-staff-engineers/9781098118723/ch03.html#iterating_on_writing_a_vision_or_strate)), maybe even occasionally dropping back to steps in the “Approach” section as your perspective changes.

There will always be more information, so notice when you start to get diminishing returns from this loop. It’s very easy for a vision or a strategy to keep dragging on, particularly if it’s not your primary project, so timebox this work and give yourself some deadlines. If you’ve set up milestones, use them as a reminder to stop iterating and wrap up. Don’t be afraid to stop, even if it’s not ‘perfect”: you can—and should—revisit the document regularly to see what context has changed. If you’ve missed something, there’ll be opportunities to add it later.



**Figure 3-2. Iterating on writing a vision or strategy.**

**Initial ideas**

Here are some questions you might ask when you’re initially thinking about creating a vision or strategy. These questions are just a starting point: there will be many stakeholders and perspectives, and this is just an exercise to help you get your thoughts in order before talking to others.

**What documents already exist?**

If there are visions or strategies that encompass yours, like company goals or values or a published product direction, you should “inherit” any constraints they’ve set. Here’s where the perspective from your locator map will come in handy. If you’re writing a wide-scale technical vision, you should know what your organization hopes to achieve in the next few years. The future that you’re envisioning should include success for those existing plans and for the technical changes that have to happen to underpin them.

If there are team-level or group-level documents at a smaller scope than yours, be aware of those too. Sometimes it’s inevitable that a vision or strategy with a broader scope will cause a narrower one to have to change, but understand the disruption that will cause and weigh it up when you’re thinking about trade-offs.

**What needs to change?**

What’s difficult right now? If your teams are complaining about being blocked by dependencies on other teams, you might want to emphasize autonomy. If new features are slow to ship, maybe what you want is fast iteration speed. If your product is down as much as it’s up, maybe you need a focus on reliability.

Knowing what you do about your company, where should your group be investing? Mark Richards and Neal Ford talk about “architectural characteristics” in software: scalability, extensibility, availability, and so on.[**9**](https://learning.oreilly.com/library/view/the-staff-engineers/9781098118723/ch03.html#ch01fn36) Which of those characteristics will you need to invest in as the business expands or changes?

Think big. If you’re working in a codebase that takes a day to build and deploy, it might be tempting to wish for incremental improvement. “This needs to take only half a day!” But go further than that. If you set a goal of 20-minute deploys, the teams pushing toward that goal have an incentive to have bigger, braver ideas. Maybe they’ll contemplate replacing the CI/CD system, or discarding a test framework that can never be compatible with that goal. Inspire people to get creative. (But, as I’ll discuss later in this chapter, don’t set goals that are impossible for your organization to achieve.)

**What’s great as it is?**

If you have snappy performance, rock-solid reliability, and a simple and clean UI, make sure your future state includes keeping the things that are working well. Maybe you’ll end up deliberately trading off some of that greatness for something else you want more, but don’t do it unconsciously.

**What’s important?**

You may be noticing a pattern here—this question has come up in every chapter so far (and it’ll come up again, too). Your vision or strategy will influence the work of many senior people. Don’t waste their time or yours on things that don’t matter. If you’re getting teams to do an expensive migration from one system to another, for example, there had better be a treasure at the end. The more effort it’s going to take, the better the treasure needs to be.

**What will Future You wish that Present You had done?**

Last one! I love the technique of envisioning a conversation with Future Me, two or three years older (and hopefully wiser). I’d ask what the world looks like, what we did, and what we wish we’d done. Which problems are getting a little worse every quarter and are going to be a real mess if ignored? Do your future self a favor if you can, and don’t ignore those. I call these favors “sending a cookie into the future”: it’s a small but heartfelt gift from your current self to your future self.

**Writing**

At the end of thinking through these questions, you may be starting to identify themes and have an opinion about what’s most pressing. This may be a good time to start a rough first draft. Be prepared to let other people change your mind, though, and maybe change it a lot. You’ll edit and iterate as you talk to other people and decide what to do next.

Writing as a group can be tricky. Here are two approaches:

*Have the leader write a first draft for discussion*

One option is that the leader of the group (again, I’m assuming that’s you) writes up a first draft for discussion. This approach is great for giving the document a consistent voice and set of concerns. Be aware, though, that this draft will inevitably be influenced by its author’s interests and affiliations. Reviewers and editors will be biased by what’s already in it, especially if the person writing it has more influence or seniority than they do.

You can help mitigate this effect by spending a lot of time talking as a group before you start to write. If you don’t feel strongly about some decisions or have even chosen arbitrarily, flag that clearly: “I rolled some dice and chose this direction. I think it’s a reasonable default if we can’t come to a decision. I bet we can do better, though.” Make absolutely sure that anyone who has more knowledge than you on this system will feel safe disagreeing with you. “Strong ideas weakly held” only works if you’re crystal clear that that’s what is happening.

*Aggregate multiple first drafts*

The other approach is that each person in the core group writes their own first draft, and then someone aggregates it afterward.[**10**](https://learning.oreilly.com/library/view/the-staff-engineers/9781098118723/ch03.html#ch01fn37) Mojtaba Hosseini, a director of engineering at Zapier, told me about a group that took this approach at a previous company. Having multiple documents was a great way to get everyone’s unbiased opinions, he said, but some participants ended up getting emotionally invested in their own document, criticizing the others instead of contributing to them. That group hadn’t nominated anyone to combine the drafts or act as a tiebreaker when two disagreed. Hosseini advises making clear up front that these documents are all inputs to one final document that everyone gets to review at the end—no one document will be the “winner.” Set expectations about who will write that final version and mediate disagreements.

**Interviews**

Your core group’s ideas and opinions will reflect only the experiences of the people in that group. You might not know what you don’t know about what’s difficult in teams other than yours—so put your preconceptions aside and talk to people. Lots of people. Don’t just pick the colleagues you already know and like: chances are they’re organizationally pretty close to you. Seek out the leaders, the influencers, and the people close to the work in other areas.

Early on, you might ask broad, open-ended questions in your interviews.

* “We’re creating a plan for X. What’s important to include?”
* What’s it like working with…?”
* If you could wave a magic wand, what would be different about…?”

When you have scoped and framed your work, you might scope the conversation by describing how you’re thinking about the topic, sharing a work-in-progress document, or asking for their reaction to a straw man approach. Optimize for getting as much useful information as possible and for making your interviewee feel like part of what you’re doing. I always end this kind of interview with “What else should I have asked you? Is there anything important I missed?”

Interviewing has another benefit: it shows the interviewees that you value their ideas and intend to include them in what you write. You’ll hear about problems that you hadn’t considered and new opinions about problems you already know about. Other people may disagree with you about what the biggest problems are. Have an open mind and take their thoughts seriously.

**Thinking time**

However you and your group like to process information (whiteboarding, writing, drawing diagrams, structured debate, sitting in silence and staring at a wall), make sure you give yourselves a lot of time to do that. I think best by writing, so when working on a vision or strategy, I need to write out my thoughts, then refine and edit them for a long time until they make more sense to me. I also get a lot from just talking through the ideas with colleagues, asking ourselves questions and trying to pick apart nuances. My colleague Carl, by contrast, likes to load up his brain with information and sleep on it: he’ll usually have new insights the next morning. In some cases, you’ll be able to build prototypes to test out your ideas. In others, the strategy will be more high-level and you’ll have to walk through the consequences as a thought experiment instead.

Be open to shifts in your thinking. As you make progress in identifying the areas of focus or the challenges to be solved, you’ll notice that you’re finding new ways to talk about them. Lean into this and help it happen. The mental models and abstractions you build will help you think bigger thoughts.

Thinking time is also a good time to check in on your motivations. Notice if you’re describing a problem in terms of a solution you’ve already chosen—this can be a mental block for a lot of engineers. We start out by comparing problems to solve, but find ourselves talking in terms of technology or architecture we “should” be using that would make everything better. As Cindy Sridharan [says](https://oreil.ly/DggW1), “A charismatic or persuasive engineer can successfully sell their passion project as one that’s beneficial to the org, when the benefits might at best be superficial.” Be especially aware of what you’re selling to yourself! When looking at the work you’re proposing to do, ask “Yes, but *why*?” over and over again until you’re *sure* the explanation maps back to the goal you’re trying to achieve. If the connection is tenuous, be honest about that. Your pet project’s time will come. This isn’t it.

**Make Decisions**

At every stage of creating a vision or a strategy, you’re going to need to make decisions. This chapter has discussed many early decisions: what kind of document to create, who to get involved, who to ask for sponsorship, how to scope your ambition, which goals or problems to focus on, who to interview, and how to frame the work. As you work through your vision or strategy, you’ll need to weigh trade-offs, decide how to solve a problem, and decide which group of people won’t get their wish.

**Trade-offs**

The reason decisions are hard is that all options have pros and cons. No matter what you choose, there will be disadvantages. Weighing your priorities in advance can help you decide which disadvantages you’re willing to accept. The same holds for advantages: every solution is probably good for something!

One of the best ways I’ve seen to clarify trade-offs is to compare two positive attributes of the outcome you want. I’ve heard these called *even over* statements. When you say “We will optimize for ease of use *even over* performance” or “We will choose long-term support *even over* time to market,” you’re saying, (to quote [“The Agile Manifesto”](https://agilemanifesto.org/)), “While there is value in the items on the right, we value the items on the left more.” This framing makes the trade-offs clear.

**Building consensus**

Sometimes none of the options on the table can make everyone happy. Do try to get aligned, but don’t block on full consensus: you might be waiting forever, and so you’re back to implicitly choosing the status quo. Take a tactic from the Internet Engineering Task Force (IETF), whose [principles](https://oreil.ly/x3Bds) of decision making famously reject “kings, presidents, and voting” in favor of what they call “rough consensus,” positing that “lack of disagreement is more important than agreement.” In other words, take the sense of the group, but don’t insist that everyone must perfectly agree. Rather than asking, “Is everyone OK with choice A?” ask, “Can anyone not live with choice A?”

When IETF working groups make decisions, they’re looking for a large majority of the group to agree and for the major dissenting points to have been addressed and debated, even if not to everyone’s satisfaction. There may not be an outcome that makes everyone happy, and they’re OK with that. [Mark Nottingham’s foreword to *Learning HTTP/2*](https://oreil.ly/lP2OO) by Stephen Ludin and Javier Garza (O’Reilly) notes of his experience in one such group: “In a few cases it was agreed that moving forward was more important than one person’s argument carrying the day, so we made decisions by flipping a coin.”

If rough consensus can’t get you to a conclusion and you’re not ready to flip a coin, someone will need to make the call. This is why it’s best to decide up front if you have, or someone else has, clear leadership authority and can act as a tiebreaker. If there’s nobody in that role, you could ask your sponsor to adjudicate, but make this a last resort. Your sponsor is likely so far away from the decision that they don’t have all of the context, you’d be asking for a lot of their time, and they might end up picking a path that nobody is happy with.

**Not deciding is a decision (just usually not a good one)**

When you’re choosing between Option A and Option B, there’s an implicit third option, C: *don’t decide*. People often default to Option C, because it lets them stay on the fence and not upset anyone. This is the *worst* thing you can do. Decisions constrain possibilities and make it possible to make progress. Not deciding is in itself a decision to maintain the status quo, as well as the uncertainty that surrounds it. While keeping your options open indefinitely might feel like it’s giving you flexibility, in the longer term your solution space stays large. Other decisions that depend on this one have to hedge to prepare for any of the possible directions you might end up choosing later.

If you realize you’ll need more information to make an informed decision, what extra information do you expect to get, and how? If you choose to wait, what are you waiting *for*? Remember that you usually don’t need to make the *best* decision, just a *good enough* decision. If you’re stuck, timebox it. Instead of saying, “We’ll choose the best storage system on the planet,” try, “Let’s research storage systems for the next two weeks and choose by the end of that time.”

There are sometimes good arguments for postponing a decision. The excellent decision-making website [The Decider](https://thedecider.app/) lists a few: when the time and energy you would need to invest in deciding just isn’t worth any of the benefits the decision will give you, when getting it wrong carries heavy penalties and getting it right carries little reward, or when you suspect the situation might just go away on its own. But the key is that you *decide* not to decide. You add “make no decision (status quo)” to your list of options and deliberately choose it. You don’t just throw up your hands and walk away.

When you can’t be sure that your decision is a great one, think about what could go wrong. Can you include ways to course-correct or mitigate any negative outcomes? Make it so that if you’re wrong, it’s not a terrible thing.

**Show your work**

However you make the decision, document it, including the trade-offs you considered and how you got to the decision in the end. Don’t elide the disadvantages: be very clear about them and explain why you’ve decided this is still the right path. In some cases, it’s genuinely going to be impossible to make everyone happy, but you can at least show that you’ve understood and considered all of the arguments. Not only is it respectful to give your coworkers this information, it also reduces the risk of having to relitigate the decision every time a new person joins the project and assumes you haven’t noticed the disadvantages they can see.

Making someone unhappy is, unfortunately, inevitable when you’re creating a strategy or vision. If everyone gets their wish, it’s unlikely that you made any real decisions. Be empathetic and try to solve everyone’s problems when you can, but make a decisive call and show why you chose what you chose.

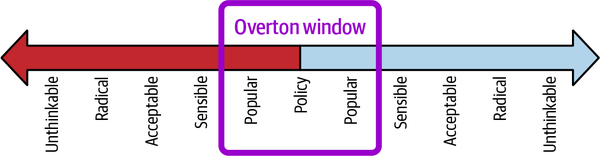
**Get Aligned and Stay Aligned**

Understand who your final audience will be. Will you need to convince a small number of fellow developers? The whole company? People outside your company? Think about how you can bring each group along on the journey with you.

Keep your sponsor up to date on what you’re planning and how it’s going. That doesn’t mean you should send them an unedited 20-page document while you’re still trying to figure out what point you’re trying to make. Take the time to get your thoughts together so that you can bring them a summary of how you’re approaching various problems and what your options are. Unless they want to see the work in progress, share the highlights of what you’re writing, rather than the gory details. In particular, if you’re writing a strategy, make sure you’re aligned *at least* at the major checkpoints: after you’ve framed the diagnosis, after you’ve chosen a guiding policy, and again after you’ve proposed some actions. If your sponsor believes you’re on the wrong path, you’ll want to find out before you spend a lot more time on it.

**Be reasonable**

Your path should be aspirational, but not impossible. Some changes are much too expensive to justify the investment. Other efforts just won’t win support in your organization. There’s a political science concept called [the Overton window](https://oreil.ly/wTFcR) ([Figure 3-3](https://learning.oreilly.com/library/view/the-staff-engineers/9781098118723/ch03.html#the_overton_window_shows_which_ideas_ar)): the range of ideas that are tolerated in public discourse without seeming too extreme, foolish, or risky. If your ideas are too futuristic for the people you need to believe in them, your colleagues will dismiss your document and you’ll lose credibility (more on that in [Chapter 4](https://learning.oreilly.com/library/view/the-staff-engineers/9781098118723/ch04.html#finite_time-id00004)). Be aware of what your organization will accept, and don’t take on an unwinnable battle.



**Figure 3-3. The Overton window shows which ideas are politically acceptable at a given time (source: based on an image from the**[**Toronto Guardian**](https://oreil.ly/dANiZ)**).**

**Nemawashi**

If you keep your stakeholders aligned as you go along, your document won’t ever have a point where you’re sharing a finished document with a group of people who are learning about it for the first time. When I spoke with Zach Millman, pillar tech lead at Asana, about creating a strategy there, he told me that he used the process of [*nemawashi*](https://oreil.ly/cu6py)*,* one of the [pillars of the Toyota Production System](https://oreil.ly/Namw7). It means sharing information and laying the foundations so that by the time a decision is made, there’s already a consensus of opinion.[**11**](https://learning.oreilly.com/library/view/the-staff-engineers/9781098118723/ch03.html#ch01fn38) If there’s someone who’ll need to give a thumbs-up to your plan, you’ll want those people to show up to any decision-making meeting already convinced that the change is the right thing to do. I’ve always framed this as “Don’t call for a vote until you know you have the votes,” but I was delighted to learn that there’s a word for it.

Keavy McMinn told a similar story of a strategy she created while she was at Github. By the time she was ready to share the document with the whole company, she had complete buy-in from her boss and his boss, and she’d done a ton of behind-the-scenes prework. The decision makers already knew that the work should be staffed. Launching the document still built momentum and excitement around the work and helped a broader audience clarify the details and buy in to the decisions.

Don’t forget that aligning doesn’t just mean convincing people of things. It goes both ways. As you discuss your plans for a vision or strategy, those plans might change. You might realize that many people are getting hung up on some aspect of your document that wasn’t really important to you, and so you end up removing it. You might compromise on some point that is a source of conflict, or give extra prominence to something that wasn’t hugely important to you but that is really resonating with your audience. You might even legitimately find a better destination to aim for. All of this is OK, and is why writing a document like this takes time.

**Work on your story**

A vision or strategy that not everyone knows is of little value to you. You’ll know the direction is well understood if people continue to stay on course when you’re not in the room to influence their decisions. But to make that happen, you’ll need to get the information into everyone’s brains. You can’t do that if you give your organization a long document to memorize; you’ll need to help them out. This is a place where the pithy one-liner or “bumper sticker” slogan I mentioned earlier can really shine. In his article [“Making the Case for Cloud Only”](https://oreil.ly/jZ9SS), Mark Barnes writes about coming up with the slogan “Cloud only 2020” as a powerful way to make it easy for everyone at the *Financial Times* to remember their cloud strategy. Sarah Wells, [speaking about the same migration](https://oreil.ly/juTzy), added, “It’s certainly the one thing from our tech strategy that developers could quote.” If your teams know, understand, and keep repeating the story of where they’re going, you’re much more likely to get there.

Your project is also more likely to be successful—and cost less social capital—if you can convince people that they *want* to go to the place you’re describing. As you write, think about how your words will be received and be clear about what story you’re trying to tell. To get back to the idea of drawing a treasure map, imagine that you’ve done that, and now you’re in the pirate bar, rolling your treasure map out on the table and trying to make the other people at your table want to come along with you. What are you telling them?

You want a story that is *comprehensible*, *relatable*, and *comfortable*.

First, you’ll want to make sure the story is comprehensible. A short, coherent story is much more compelling than a list of unconnected tasks. It’s hard to make people enthusiastic about something they don’t understand, and you’re missing the opportunity to have them tell the story when you’re not there. Even if they’re brought along by your enthusiasm, if they don’t really understand the plan, they can’t champion it.

Make sure the story is relatable. The reason the treasure is exciting for you might not be at all exciting for other people, so the way you frame the story really matters. If your vision is that your own team will have solved its most annoying problems, live happier lives, and eat ice cream, that’s pretty compelling…for people on your team. If achieving that vision will need work from other teams, you’ll need more. Show how your work will make their lives better too.

Similarly, remember the Overton window and make sure the story is comfortable. A compelling story to take people on a journey from A to B will only work if they’re actually at A. If they’re a long distance back from there, you might have more success in convincing them of A, and then waiting until that idea is considered sensible and well-accepted before taking them on the next step.

Your story will help people as they execute on the plan too. As Mojtaba Hosseini told me, “When it gets difficult, everyone needs to know that the difficulties are expected, and that they can be overcome. Don’t just tell the story of the gold at the end of the journey. When there are problems, you need to be able to emphasize that this is the part of the story where the heroes get caught in the pit…but then they get out again!”

**Create the Final Draft**

It can feel hard to believe, when you’ve spent weeks or months creating a document, but not everyone will be excited to review it. Don’t be offended! While people may have the best intentions, a lengthy document can stay open in a tab for a long, long time. Think about how you can make it easy to read, or share the highlights in another way. Avoid dense walls of text. Use images, bullet points, and lots of white space. If you can find a way to make your points clear and memorable, more people will grasp them.

One way is to use “personas”: describe some of the people affected by your vision or strategy—developers, end users, or whoever your stakeholders are—and describe their experience before and after the work is done. Another approach is to describe a real scenario that’s difficult, expensive, or even impossible for the business now and show how that will change. Be as concrete as you can. Unless you’re presenting solely to engineers in the same domain, avoid jargon. Some of your readers will start tuning out after they hit a few acronyms or technical terms they don’t know.

You may find it useful to have a second type of document to accompany the one that you’ve written. If you’re going to present at an all-hands meeting or similar, you’ll want a slide deck. You may want both a detailed essay-style or bullet-pointed document and also a one-page elevator pitch with the high-level ideas. If you’re comfortable sharing with people outside the company, you might write an external blog post: this can be another opportunity to reach internal audiences too.

**The Launch**

There’s a difference between a vision document that is one person’s idea and a vision that is the company or organization’s officially endorsed north star, with teams working to achieve it. I have seen so many documents die at this point because the authors didn’t know how to make them real.

**Make It Official**

What’s the difference between your document being *yours* and *the organization’s*? Belief, mostly. That starts with endorsement from whoever is the ultimate authority for whatever scope your document needs. Usually this is whoever is at the top of the people-manager chain: your director, VPs, CTO, or other executive. If you’ve been using nemawashi and staying aligned with the people whose opinions matter, that person might already be on board. If so, see if they’re willing to send an email, add their name to the document as an endorsement, refer to it when describing the next quarter’s goals, invite you to present your plan at an appropriately sized all-hands, or make some other public gesture of accepting the plan as real. If you don’t have their support, ask your sponsor to help you sell the idea.

Make sure your document looks real. Host it on an official-looking internal website. Close any remaining comments and remove any to-dos. Consider removing the ability to add comments, and leave a contact address for feedback instead. If you can include the head of the department or similar as a contact, that’ll carry a lot more weight than if it has just engineers’ names on top.

An officially endorsed document gives people a tool they can use for making decisions. However, there’s another important part of making the document real: actually staffing the work in it. If you’ve proposed new projects or cross-organization work, you may need headcount—and actual humans to fill that headcount. If you’ll need a budget, computing power, or other resources to make the work happen, that need should have come up in the course of agreeing on the direction, but now you’ll face the reality of actually getting it. Talk with your sponsor about how to work within your regular prioritization, headcount, OKR, or budgetary processes.

Depending on your organization, you may be personally responsible for starting to execute on the strategy, or you may be handing it off to other people to make the work happen. In my experience, you’ll all be more successful if you stay with it, making sure the work maintains momentum and the plan stays clear as the vision or strategy turns into actual projects.

**Keep It Fresh**

Shipping a vision or strategy doesn’t mean you can stop thinking about it. The business direction or broader technical context may change and you’ll need to adapt. You may also just find out that the direction you chose was wrong. It happens. Be prepared to revisit your document in a year, or earlier if you realize that it’s not working. If the vision or strategy is no longer solving your business problems, don’t be afraid to iterate on it. Explain what new information you have or what’s changed, update it, and tell a new story.

**Case Study: SockMatcher**

Time to go back to our friends at SockMatcher. When you read the scenario at the start of the chapter, maybe you had ideas for what they should do. In some ways, the technical problems are easy to solve. But making a change that affects many people is anything but.

Let’s imagine you’re a staff engineer working at SockMatcher. Here’s the story of your approach, writing, and launch.

**Approach**

Your previous project has just wrapped up and you’re looking for something impactful to do next, ideally something that’s a bit of a stretch. Creating a plan for the most contested core architecture in the company certainly fits the bill. It also feels like an important problem, one that can have a huge impact on the business.

Your manager is wary. Many others have tried to tackle this architecture before—this may be an uncrossable desert! You suggest that you take a couple of weeks to understand why previous attempts failed. If you don’t have a compelling reason to believe that your journey can be different, you won’t do it.

**Why didn’t previous attempts work?**

You start out by chatting with two staff engineers who have taken a run at rearchitecting the monolith in the past.

The first, Pierre, spent three months creating a detailed technical design for the monolith and surrounding architecture. Other teams weren’t impressed: they disagreed with some trade-offs Pierre had made, the direction didn’t match their own plans for their components, and they didn’t like having a solution handed to them to implement. Unable to rally enthusiasm, Pierre decided the project couldn’t be solved (at least, not with the current set of engineers). He’s still pretty grumpy about it.

The other engineer, Geneva, set out to build a coalition before attempting a rearchitecture. She set up a working group and there was a ton of interest. The various participants were initially eager to work together, but the working group got bogged down in debate and couldn’t agree on a path. The hours of meetings became a time sink and people stopped going, including Geneva.

As you talk with these and other engineers who have opinions about “solving” the monolith, you notice two patterns. The first is that most people have a specific solution in mind: “The problem is that we don’t have microservices” or “We just need to shard the data stores.” Everyone wants their solution to “win,” so consensus is impossible. The second pattern is that everyone is focused only on the technical problems. There are lots of technically sound ideas, but no plans for how to get the organization to buy in to a path forward.

You think you can be more successful if you tackle organizational alignment as the crux of the problem. You resolve to get an executive sponsor and make sure that any directions you propose are not just good technical solutions but are viable within *this* organization. You’ll be pragmatic and low-ego, helping existing ideas succeed rather than trying to have your own direction prevail.

**Sponsorship**

You’ve got some social capital and credibility (see [Chapter 4](https://learning.oreilly.com/library/view/the-staff-engineers/9781098118723/ch04.html#finite_time-id00004)) in the bank after your last project, but you know that’s not enough to convince all of the many teams that care about the monolith. You also know that you’re going to have to make some decisions and you won’t be able to make *everyone* happy. If you need complete consensus, you’re not going to succeed. Also, any plan you make is likely to create engineering projects. If you’re not going to be able to staff the work, you’d rather find that out early, before you waste your time. You need an executive sponsor.

You start with Jody, the director whose teams operate the monolith: she has a vested interest in making it easier to maintain. But she’s seen her people get pulled into the two previous attempts to change the architecture, and she wants to defend their time. They have their own projects, and she doesn’t want them to be distracted by yet another new initiative. While she’s in favor of rearchitecture, it’s in a “Next year, we hope, maybe” sort of way. She’s not interested in committing anyone to this work.

Your next stop is Jesse, the director who’s taking on the food storage container launch. With this high-profile project coming online, Jesse might have easier access to staffing and, if you can align his success metrics with your own, he’s likely to support the work. When you talk to Jesse, you describe a future where product teams can work autonomously, product engineers are happier, and new features are delivered quickly. Jesse isn’t sure. That’s a nice future, but food storage needs to launch this year; they can’t wait for a massive rearchitecture. You agree: any solution must let the food storage folks launch with minimal friction. Jesse is convinced. He agrees to sponsor and support your work.

**Other engineers**

You look for coauthors, a few colleagues who can bring different perspectives and knowledge to the work. You also want to build allies across your organization and engage anyone who will be skeptical of or pull against your plan.

You start with Pierre, the staff engineer who proposed the detailed previous solution. He’s still feeling a bit raw from putting his heart into making a thorough solution and meeting with complete apathy. He says some defeated (and kind of mean) things about the company leadership and makes it clear that he thinks your work is a waste of time. You ask if you can use his previous plan as an input to some of the work you’re doing, crediting him for any parts of his work you end up using—though you set expectations that you’ll scope your project differently. The idea of his work getting used makes him a little more willing to help. He still won’t join your group, but he agrees to be interviewed and to review your plans later.

You have higher hopes as you invite Geneva to join efforts. She’s in. You’re surprised when she tells you that her previous working group still exists, sort of: three senior engineers meet and talk about architecture every week. They’ve given the monolith problem a ton of thought, and you know they’ll be able to see nuances that wouldn’t be immediately obvious to you. You invite them to team up, asking them to commit two days a week for at least two months. One engineer, Fran, agrees; the other two want to advise but can’t commit a big block of time. You agree to come back to the working group meeting every few weeks with updates.

You check in with some other potential allies:

* The team lead on the food storage container project is inclined to see problems as easier to solve than they actually are, especially when it comes to work assigned to the monolith maintenance team. “Why don’t they just build isolated modules within the monolith?” she asks. But if there’s a plan, she’s on board.
* The databases lead is wary that your project will drop unexpected work on his team. (It’s justifiable: there’s history.) You promise to keep him in the loop and let him review plans early on.
* The staff engineer who wrote the original sock matching code is very tenured and *very* influential. If she’s convinced, a lot of other people will be too. She has some ideas and wants to be an early reviewer, too.

**Scope**

Your core group—you, Geneva, and Fran—talk about what you want to do and what might be successful. Fran is eager to create a technical vision for how all of your architecture evolves, but it’s not the right choice for your situation: you don’t have that scope of influence and neither does your sponsor. Also, a project of that scope couldn’t be ready in time for the food storage launch.

What about a vision for the core monolith architecture? That would clarify where you’re all going, but teams would still be divided on how to get there. You decide you need a broad architectural plan for the monolith and a strategy for how to get there that explicitly includes the food storage launch. You’ll aim to describe one year’s work and stay at a high level. You commit to making occasional lower-level technical decisions, but only when the decision doesn’t have an owner: you’ll leave most of the implementation details to the teams that will do the work. And your strategy has to be official: it can’t be just *a* plan; it has to be the organization’s chosen plan or you won’t consider the project a success.

Once you’re all on the same page, you write down what you’re going to do: “Create a high-level one-year technical strategy for enabling the food storage launch while evolving our core monolith architecture.” It’s a little vague, but it’s a start!

You talk more about the scope and the problems, collecting links to the previous efforts and the working group’s notes, and getting yourselves (and your shared vocabulary) aligned. Your document is rough and not something you’d share outside the core group, but it keeps your ideas in one place and lets you all add extra thoughts as they come to you.

Once you have an elevator pitch for what you’re aiming to do, you check in with Jesse, your sponsor. He’s on board with the scope and the kind of document you’re creating. His suggestion for making your plan official is to add an organizational OKR for it, with your name as the directly responsible individual (DRI). That’s a little intimidating, but it will certainly give you the official endorsement you were hoping for. He offers to make sure Jody and the other directors are comfortable with it, and to get the OKR added.

You create a discussion channel for the effort, announce it in other channels that are likely to have interested parties, and share notes about your scope and what prior art you’re drawing from. You highlight that you want to talk with people who have opinions, and are still welcoming collaborators who have at least two days a week to spend on it. There are a *lot* of the former—and none of the latter. You start listing people to talk with and set out to write your strategy.

**The Writing**

Before you jump into solutions, you want to be *really* clear about what problems you’re solving. Your initial scope of “Create a high-level one-year technical strategy for enabling the food storage launch while evolving our core monolith architecture” needs more clarity, but you don’t want to jump to solutioning either. You need to diagnose the situation and describe exactly what’s going on.

**Diagnosis**

There are so many facts that you could consider:

* There’s an immediate product need to support food storage containers, and there are indications that product lines will expand more in future.
* The team running the monolith is getting paged too much: that’s not sustainable and needs to change.
* Your matching algorithm is a little slow and could have a higher hit rate.
* Your systems are not currently handling spikes in traffic.
* Users are unhappy with your availability.
* The login system is old and has a lot of technical debt.
* Deploying new code is slow and frustrating.
* External users depend on APIs you wish you could change.
* Adding new products to match requires major logic changes in several core components.
* Many developers just don’t like working in the monolith.

And that’s not even the entire list! You set out to select the facts that matter most and tell a much simpler story. Your group spends some time brainstorming about what’s important, what’s not working, and what’s great as it is. You imagine your future selves looking at the same codebase with twice as many engineers and another five products. If those products were implemented as edge cases too, navigating the business logic for each feature would become horrific, and changing anything in that scenario would be complicated and fraught. Builds would be slower and deploys would fail more often. More celebrity users would mean more outages. If you take no action, this is the future. You’d better take some action!

Although you’ve got lots of ideas after your brainstorming session, you don’t want to get too committed to them just yet. Instead, you go chat with some members of your product teams, as well as engineering leaders and practitioners, aiming to understand what’s important to them. You learn some new information:

* While you’d speculated that traffic spikes from celebrity endorsements might be causing the decrease in availability, outages caused by overloads are actually uncommon and brief: just a few minutes of downtime per month. While you should certainly improve here, these outages are not the most important contributor to your poor availability.[**12**](https://learning.oreilly.com/library/view/the-staff-engineers/9781098118723/ch03.html#ch01fn39) It turns out that the real damage is being caused by unremarkable code bugs, and by the fact that it takes three hours to deploy a fix. Previous attempts to improve reliability have centered on adding more testing to your release path—but this actually slowed deploy times and lengthened outages.
* Many engineering teams complain about working in the monolith, but the thing they actually hate is releasing code. A huge percentage of changes have unexpected behavior. It’s hard to be sure that your change isn’t breaking someone else, and getting a fix out takes half a day. Teams are twitchy from responding to outages.
* The billing and personalization subsystems are by far the most contested parts of the codebase. Most major feature changes come with a corresponding change in one or both of those pieces of functionality, and their logic is so complex that it’s easy to have unexpected side effects while making even simple changes.

You learn a lot more, too: everyone you talk to has a different topic they want to tell you about. But the interviews build up a pattern and let you see what’s going on. You choose the most important subset of the facts and tell a much simpler story.

Here’s your diagnosis:

*Every new feature change needs complex logic changes in a set of shared components. Modifying these components is slow and difficult. Unexpected interactions mean that teams are constantly disrupting each other’s work and causing long outages. Every new type of matching item will increase the number of points of coupling inside these shared components and make the problem worse. Our systems need to be able to handle more matching components and more teams adding them without development grinding to a halt.*

Choosing your focus can be one of the most painful parts of writing a strategy. In this case, the unversioned APIs are still a problem, the unpleasant login code will become a problem eventually, and improving the match functionality is a real opportunity that you’re not going to try for right now. Those problems and opportunities are real, but you’re making a hard decision and ignoring them for now.

With the diagnosis, you check in with your sponsor, Jesse, and make sure he agrees that you’re focusing in the right area. You show him the list of challenges you’re *not* focusing on too, to make it clear that you see them but don’t think they should come first. Jesse agrees.

**Guiding policy**

Now that you’ve got a clear sense of what’s happening, you can decide what to do about it.

There’s a proposed guiding policy already on the table: some of your colleagues are pushing to completely break down the monolith. They say that the rearchitecture would make it easier to add new products, as well as reduce the number of unintended breakages and the time to get a code fix built and deployed when something’s broken. But teams would still need to make risky changes to shared components. It would also mean that all of the teams would begin running their own services, putting some of them on call for the first time in their lives. Finally, a change like that would also take at least three years, with no solution for shipping products in the meantime. So, while “let’s run microservices instead” might be the perfect solution for a company with different constraints, it doesn’t acknowledge the current situation.

Instead, you look at the places where a small amount of work can have an outsize impact. An obvious point of leverage is the two key shared components where integration slows teams: billing and personalization. If those two were easy to add to, instead of having hardcoded logic for each product, other teams could safely add new kinds of matchable items. There would be fewer outages, freeing up teams to spend their time on feature work and improving the system further. It’s a virtuous cycle.

Here’s your guiding policy:

*The billing and personalization systems should be easy and safe to integrate with.*

You write up some notes about the guiding policies you *didn’t* choose, too. You describe the reasons you considered microservices, and the advantages that path would bring, but explain why they wouldn’t solve your problems.

**Actions**

You set out to outline the actions your group will need to take to navigate the challenges and carry out your guiding policy.

It’s important that your actions are realistic, so you check in with the billing and personalization teams and leadership and confirm that they’re on board with your direction. The billing team already had some backlog work around offering a menu of billing functionality that other teams could choose by setting configuration options. The personalization folks had toyed with the idea of a plug-in architecture with stable core functionality and isolated logic for each type of matchable item. Teams adding new items would only have to modify their own plug-in component. Both of these changes would make the shared components more modular and enable self-service access to them, and the teams would be happy for a reason to spend time on them.

There’s a bootstrapping problem, though: these are big, risky changes. Refactoring these core components is likely to cause many outages, so the teams have not prioritized the work. You suggest adding the ability to release changes behind [feature flags](https://oreil.ly/ErwHv), so a regression can be a quick switch back, rather than another deploy. Safer deploys would reduce the cost and risk of the work.

The isolation work won’t happen overnight, and the food storage team will need integrations with both these components in the meantime. The personalization and billing teams are willing to treat food storage as a pilot customer and optimize for making them successful, including writing the first version of their integrations in their existing systems and migrating them to the self-service model when it’s available. But those teams can’t take on both isolation and integration work at once with their current staffing. Jesse agrees to donate some of the headcount that had been allocated to the food storage project and let both those teams grow.

Here are your actions:

* Add a feature flagging system that allows staged rollouts and quick rollbacks.
* Add two engineers to the payment team and one to the personalization team.
* Modify the billing and personalization subsystems to allow easy, safe, self-service additions of new matchable items.
* Have the billing and personalization teams onboard the food storage product into their systems, then migrate them to be pilot customers of the new self-service approach.

These actions are high-level and the teams involved have autonomy to design solutions and make a lot of decisions. But this approach gives them a direction and some concrete next steps. There are many, many other suggestions for actions: everyone you talk to has a laundry list of the work they think should happen to make the monolith healthier. But your guiding policy lets you focus your efforts and keep your list short.

You nominate Fran as the primary author to write up the plan, with you and Geneva making many suggested edits. The document is honest about the trade-offs of your plan as well as the alternatives you considered and why you didn’t choose them.

After you’ve aligned with your sponsor (he suggests some wording changes but is overall enthusiastic), you road test your plan by sharing the first draft with a few of your allies. Some leave comments. You interview others in person and shake out some concerns.

Your story becomes a little tighter every time you tell it. You share the document with progressively broader groups and start to present about it at some meetings.

**The Launch**

Let’s be honest: your plan is not universally beloved. Some of your colleagues are underwhelmed (and, perhaps, a little angry): your document isn’t more “visionary” than any of the ideas they had; it’s a little boring! Some insist that this problem didn’t need a strategy; you just needed to “just decide” what to do, and this path was the obvious decision.

Other vocal factions disagree that your path is obvious: they think it’s just *wrong*. One group is still arguing for moving everything to microservices. Another wants more focus on handling load spikes. And while the food storage container team can build some of their functionality as a microservice, they’ll still be developing heavily inside the monolith—some of them are unhappy about that. However, since you took the time to document the known disadvantages and the alternatives you considered, none of this is news. The grumbling doesn’t change the plan.

Happily, the positive voices are louder. Most people are energized by having a single, official, agreed-upon direction. You have particularly strong buy-in from allies who were along on the journey from the start. Your engineering-wide OKR has raised your visibility, and your sponsor and his peers are aligned with the work and willing to staff it. Your plan will take some load off the monolith maintenance team without needing a huge commitment from them, so Jody unexpectedly offers some of their help: they’ll provide the feature flag system.

You stay with the project for the next year, working directly on the billing modularization project and acting as an adviser for the personalization work. Immediately after the food storage team celebrates a successful launch, the product team announces a new effort to match missing board game pieces. Your work means that the board games team will be able to use the new isolated functionality and just start building. With stable systems, the monolith maintenance team is no longer reacting constantly: they’ve begun to work on improving the developer experience and making the systems more resilient to load spikes.

By focusing your efforts, you removed an impediment for growth, and a barrier that was slowing everyone down. And you’re ready for another project.

Onward to [Part II](https://learning.oreilly.com/library/view/the-staff-engineers/9781098118723/part02.html#part2) of this book, Execution.

**To Recap**

* A technical vision describes a future state. A technical strategy describes a plan of action.
* A document like this is usually a group effort. Although the core group creating it will usually be small, you’ll also want information, opinions, and general goodwill from a wider group.
* Have a plan up front for how to make the document become real. That usually means having an executive as a sponsor.
* Be deliberate about agreeing on a document type and a scope for the work.
* Writing the document will involve many iterations of talking to other people, refining your ideas, making decisions, writing, and realigning. It will take time.
* Your vision or strategy is only as good as the story you can tell about it.

[**1**](https://learning.oreilly.com/library/view/the-staff-engineers/9781098118723/ch03.html#ch01fn28-marker) We’re focusing on vision and strategy, but these techniques can work for any big group decisions: engineering values, coding standards, cross-organization project plans, etc.

[**2**](https://learning.oreilly.com/library/view/the-staff-engineers/9781098118723/ch03.html#ch01fn29-marker) If anyone wants to talk seed funding, drop me a line.

[**3**](https://learning.oreilly.com/library/view/the-staff-engineers/9781098118723/ch03.html#ch01fn30-marker) You might notice that these kinds of topics still consume a ton of discussion time in code and design reviews, even though they’re not at the core of any particular change. If that’s happening, or if you’re seeing designs that have *contradictory* baked-in assumptions, that’s a sign that you need to make some big central decisions, independent of any particular project or launch.

[**4**](https://learning.oreilly.com/library/view/the-staff-engineers/9781098118723/ch03.html#ch01fn31-marker) I’ll talk about these kinds of documents when we’re looking at project execution in [Chapter 5](https://learning.oreilly.com/library/view/the-staff-engineers/9781098118723/ch05.html#leading_big_projects).

[**5**](https://learning.oreilly.com/library/view/the-staff-engineers/9781098118723/ch03.html#ch01fn32-marker) That said, Patrick Shields, a staff engineer at Stripe, told me once that he encourages people to write small strategies for all sorts of things because “you need to learn to play amateur basketball before you drop into the NBA.” It’s an excellent point.

[**6**](https://learning.oreilly.com/library/view/the-staff-engineers/9781098118723/ch03.html#ch01fn33-marker) A caveat: if you’re someone who *always* takes the back seat and cheers on others, make sure your organization recognizes that leadership. If not, make sure you have some opportunities to shine too. I’ll talk about credibility and social capital in [Chapter 4](https://learning.oreilly.com/library/view/the-staff-engineers/9781098118723/ch04.html#finite_time-id00004).

[**7**](https://learning.oreilly.com/library/view/the-staff-engineers/9781098118723/ch03.html#ch01fn34-marker) If you jumped ahead to this strategy chapter and skipped all of that “What even is your job?” introspection earlier in the book, your *scope* is the domain, team, or teams that you feel responsible for. It often covers the same area that your manager covers, but not always.

[**8**](https://learning.oreilly.com/library/view/the-staff-engineers/9781098118723/ch03.html#ch01fn35-marker) The answer to this one is usually no.

[**9**](https://learning.oreilly.com/library/view/the-staff-engineers/9781098118723/ch03.html#ch01fn36-marker) They’ve got a thorough list in Chapter 4 of their book, [*Fundamentals of Software Architecture*](https://oreil.ly/RLl0h) (O’Reilly).

[**10**](https://learning.oreilly.com/library/view/the-staff-engineers/9781098118723/ch03.html#ch01fn37-marker) Sarah Grey, development editor for this book, says that this could be an editorial nightmare if not handled carefully and that the thought of aggregating all these drafts gives her a headache. If you take this path, know what you’re getting into.

[**11**](https://learning.oreilly.com/library/view/the-staff-engineers/9781098118723/ch03.html#ch01fn38-marker) As he [writes](https://oreil.ly/zJhkc), you want to hear the “spicy opinions” in one-on-one meetings and adjust your plan as necessary before the decision makers meet as a group.

[**12**](https://learning.oreilly.com/library/view/the-staff-engineers/9781098118723/ch03.html#ch01fn39-marker) If dropping some of this celebrity traffic was considered to be bad PR or a missed opportunity, you might prioritize it anyway. Context is everything.