Chapter 5. How to Lead a Team

Written by Brian Fitzpatrick

Edited by Riona MacNamara

We've covered a lot of ground so far on the culture and composition of teams writing software, and in this chapter, we'll take a look at the person ultimately responsible for making it all work.

No team can function well without a leader, especially at Google where engineering is almost exclusively a team endeavor. At Google, we recognize two different leadership roles. A "Manager" is a leader of people, whereas while a "Tech Lead" leads technology efforts. Although the responsibilities of these two roles are quite different, they require quite similar skills.

A boat without a captain is nothing more than a floating waiting room: unless someone grabs the rudder and starts the engine, it's just going to drift along aimlessly with the current. A piece of software is just like that boat: if no one pilots it, you're left with a group of engineers burning up valuable time, just sitting around waiting for something to happen (or worse, still writing code that you don't need). Although this chapter is about people management and technical leadership, it is still worth a read if you're an individual contributor because it will likely help you understand your own leaders a bit better.

Managers and Tech Leads (and Both)

Whereas every engineering team generally has a leader, they acquire those leaders in different ways. This is certainly true at Google; sometimes an experienced manager comes in to run a team, and sometimes an individual contributor is promoted into a leadership position (usually of a smaller team).

In nascent teams, both roles will sometimes be filled by the same person: a "Tech Lead Manager" (TLM). On larger teams, an experienced people manager will step in to take on the management role, whereas a senior engineer with extensive experience will step into the tech lead role. Even though manager and tech lead each play an important part in the growth and productivity of an engineering team, the skills required to succeed in each role are wildly different.

The Engineering Manager

Many companies bring in trained people managers who might know little to nothing about software engineering to run their engineering teams. Google decided early on, however, that its software engineering managers should have an engineering background. This meant hiring experienced managers who used to be software engineers, or training software engineers to be managers (more on this later).

At the highest level, an engineering manager is responsible for the performance, productivity, and happiness of every person on their team—including their tech lead—while still making sure that the needs of the business are met with the product for which they are responsible. Because the needs of the business and the needs of individual team members don't always align, this can often place a manager in a difficult position.

The Tech Lead

The tech lead (TL) of a team—who will often report to the manager of that team—is responsible for (surprise!) the technical aspects of the product, including technology decisions and choices, architecture, priorities, velocity, and general project management (although on larger teams they might have program managers helping out with this). The TL will usually work hand in hand with the engineering manager to ensure that the team is adequately staffed for their product and that engineers are set to work on tasks that best match their skillset and skill level. Most TLs are also individual contributors, which often forces them to choose between doing something quickly themselves or delegating it to a team member to do (sometimes) more slowly. The latter is most often the correct decision for the TL as they grow the size and capability of their team.

The Tech Lead Manager

On small and nascent teams, for which engineering managers need a strong technical skillset, the default is often to have a tech lead manager (TLM): a single person who can handle both the people and technical needs of their team. Sometimes, a TLM is a more senior person, but more often than not, the role is taken on by someone who was, until recently, an individual contributor.

At Google, it's customary for larger, well-established teams to have a pair of leaders, one TL and one engineering manager, working together as partners.

The theory is that it's really difficult to do both jobs at the same time (well) without completely burning out, so it's better to have two specialists crushing each role with dedicated focus.

The job of TLM is a tricky one, and often requires the TLM to learn how to balance individual work, delegation, and people management. As such, it usually requires a high degree of mentoring and assistance from more experienced TLMs (in fact, we recommend that in addition to taking a number of classes that Google offers on this subject, a newly minted TLM seek out a senior mentor who can advise them regularly as they grow into the role).

Case Study: Influencing Without Authority

It's generally accepted that you can get folks who report to you to do the work that you need done for your products, but it's different when you need to get people outside of your organization—or heck, even outside of your product area sometimes—to do something that you think needs to be done. This "influence without authority" is one of the most powerful leadership traits that you can develop.

For example, for years Jeff Dean, senior engineering fellow and possibly the most well-known Googler *inside* of Google, led only a fraction of Google's engineering team, but his influence on technical decisions and direction reaches to the ends of the entire engineering organization and beyond (thanks to his writing and speaking outside of the company).

Another example is a team that I started called The Data Liberation Front: with a team of less than a half-dozen engineers, we managed to get more than 50 Google products to export their data through a product that we launched called Google Takeout. At the time, there was no formal directive from the executive level at Google for all products to be a part of Takeout, so how did we get hundreds of engineers to contribute to this effort? By identifying a strategic need for the company, showing how it linked to the mission and existing priorities of the company, and working with a small group of engineers to develop a tool that allowed teams to quickly and easily integrate with Takeout.

Moving from an Individual Contributor Role to a Leadership Role

Whether or not they're officially appointed, someone needs to get into the driver's seat if your product is ever going to go anywhere, and if you're the motivated, impatient type, that person might be you. You might find yourself sucked into helping your team resolve conflicts, make decisions, and coordinate people. It happens all the time, and often by accident. Maybe you never intended to become a "leader," but somehow it happened anyway. Some people refer to this affliction as "manageritis."

Even if you've sworn to yourself that you'll never become a manager, at some point in your career you're likely to find yourself in a leadership position, especially if you've been successful in your role. The rest of this chapter is intended to help you understand what to do when this happens.

We're not here to attempt to convince you to become a manager, but rather to help show why the best leaders work to serve their team using the principles of humility, respect, and trust. Understanding the ins and outs of leadership is a vital skill for influencing the direction of your work. If you want to steer the boat for your project and not just go along for the ride, you need to know how to navigate or you'll run yourself (and your project) onto a sandbar.

The Only Thing to Fear Is…Well, Everything

Aside from the general sense of malaise that most people feel when they hear the word "manager," there are a number of reasons that most people don't want to become managers. The biggest reason you'll hear in the software development world is that you spend much less time writing code. This is true whether you become a TL or an engineering manager, and I'll talk more about this later in the chapter, but first, let's cover some more reasons why most of us avoid becoming managers.

If you've spent the majority of your career writing code, you typically end a day with something you can point to—whether it's code, a design document, or a pile of bugs you just closed—and say, "That's what I did today." But at the end of a busy day of "management" you'll usually find yourself thinking, "I didn't do a damned thing today." It's the equivalent of spending years counting the number of apples you picked each day, and changing to a job growing bananas, only to say to yourself at the end of each day, "I didn't pick any apples," happily ignoring the flourishing banana trees sitting next to you. Quantifying management work is more difficult than counting widgets you turned out, but just making it possible for your team to be happy and productive is a big measure of your job. Just don't fall into the trap of counting apples when you're growing bananas.[1]

Another big reason for not becoming a manager is often unspoken but rooted in the famous "Peter Principle," which states that "In a hierarchy every employee tends to rise to his level of incompetence." Google generally avoids this by requiring that a person perform the job *above* their current level for a period of time (i.e., to "exceeds expectations" at their current level) before being promoted to that level. Most people have had a manager who was incapable of doing their job or was just really bad at managing people,[2] and we know some people who have worked only for bad managers. If you've been exposed only to crappy managers for your entire career, why would you *ever* want to be a manager? Why would you want to be promoted to a role that you don't feel able to do?

There are, however, great reasons to consider becoming a TL or manager. First, it's a way to scale yourself. Even if you're great at writing code, there's still an upper limit to the amount of code you can write. Imagine how much code a team of great engineers could write under your leadership! Second, you might just be really good at it—many people who find themselves sucked into the leadership vacuum of a project discover that they're exceptionally skilled at providing the kind of guidance, help, and air cover a team or a company needs. Someone has to lead, so why not you?

Servant Leadership

There seems to be a sort of disease that strikes managers in which they forget about all the awful things their managers did to them and suddenly begin doing these same things to "manage" the people that report to them. The symptoms of this disease include, but are by no means limited to, micromanaging, ignoring low performers, and hiring pushovers. Without prompt treatment, this disease can kill an entire team. The best advice I received when I first became a manager at Google was from Steve Vinter, an engineering director at the time. He said, "Above all, resist the urge to manage." One of the greatest urges of the newly minted manager is to actively "manage" their employees because that's what a manager does, right? This typically has disastrous consequences.

The cure for the "management" disease is a liberal application of "servant leadership," which is a nice way of saying the most important thing you can do as a leader is to serve your team, much like a butler or majordomo tends to the health and well-being of a household. As a servant leader, you should strive to create an atmosphere of humility, respect, and trust. This might mean removing bureaucratic obstacles that a team member can't remove by themselves, helping a team achieve consensus, or even buying dinner for the

team when they're working late at the office. The servant leader fills in the cracks to smooth the way for their team and advises them when necessary, but still isn't afraid of getting their hands dirty. The only managing that a servant leader does is to manage both the technical and social health of the team; as tempting as it might be to focus on purely the technical health of the team, the social health of the team is just as important (but often infinitely more difficult to manage).

The Engineering Manager

So, what is actually expected of a manager at a modern software company? Before the computing age, "management" and "labor" might have taken on almost antagonistic roles, with the manager wielding all of the power and labor requiring collective action to achieve its own ends. But that isn't how modern software companies work.

Manager Is a Four-Letter Word

Before talking about the core responsibilities of an engineering manager at Google, let's review the history of managers. The present-day concept of the pointy-haired manager is partially a carryover, first from military hierarchy and later adopted by the Industrial Revolution—more than 100 years ago! Factories began popping up everywhere, and they required (usually unskilled) workers to keep the machines going. Consequently, these workers required supervisors to manage them, and because it was easy to replace these workers with other people who were desperate for a job, the managers had little motivation to treat their employees well or improve conditions for them. Whether humane or not, this method worked well for many years when the employees had nothing more to do than perform rote tasks.

Managers frequently treated employees in the same way that cart drivers would treat their mules: they motivated them by alternately leading them forward with a carrot, and, when that didn't work, whipping them with a stick. This carrot-and-stick method of management survived the transition from the factory[3] to the modern office, where the stereotype of the tough-as-nails manager-as-mule-driver flourished in the middle part of the twentieth century when employees would work at the same job for years and years.

This continues today in some industries—even in industries that require creative thinking and problem solving—despite numerous studies suggesting that the anachronistic carrot and stick is ineffective and harmful to the

productivity of creative people. Whereas the assembly-line worker of years past could be trained in days and replaced at will, software engineers working on large codebases can take months to get up to speed on a new team. Unlike the replaceable assembly-line worker, these people need nurturing, time, and space to think and create.

Today's Engineering Manager

Most people still use the title "manager" despite the fact that it's often an anachronism. The title itself often encourages new managers to *manage* their reports. Managers can wind up acting like parents,[4] and consequently employees react like children. To frame this in the context of humility, respect, and trust: if a manager makes it obvious that they trust their employee, the employee feels positive pressure to live up to that trust. It's that simple. A good manager forges the way for a team, looking out for their safety and well-being, all while making sure their needs are met. If there's one thing you remember from this chapter, make it this:

Traditional managers worry about how to get things done, whereas great managers worry about what things get done…(and trust their team to figure out how to do it).

A new engineer, Jerry, joined my team a few years ago. Jerry's last manager (at a different company) was adamant that he be at his desk from 9:00 to 5:00 every day, and assumed that if he wasn't there, he wasn't working enough (which is, of course, a ridiculous assumption). On his first day working with me, Jerry came to me at 4:40 p.m. and stammered out an apology that he had to leave 15 minutes early because he had an appointment that he had been unable to reschedule. I looked at him, smiled, and told him flat out, "Look, as long as you get your job done, I don't care what time you leave the office." Jerry stared blankly at me for a few seconds, nodded, and went on his way. I treated Jerry like an adult; he always got his work done, and I never had to worry about him being at his desk, because he didn't need a babysitter to get his work done. If your employees are so uninterested in their job that they actually need traditional-manager babysitting to be convinced to work, *that* is your real problem.

Failure Is an Option

Another way to catalyze your team is to make them feel safe and secure so that they can take greater risks by building psychological safety—meaning that your team members feel like they can be themselves without fear of negative repercussions from you or their team members. Risk is a fascinating

thing; most humans are terrible at evaluating risk, and most companies try to avoid risk at all costs. As a result of this, the usual *modus operandi* is to work conservatively and focus on smaller successes even when taking a bigger risk might mean exponentially greater success. A common saying at Google is that if you try to achieve an impossible goal, there's a good chance you'll fail, but if you fail trying to achieve the impossible, you'll most likely accomplish far more than you would have accomplished had you merely attempted something you knew you could complete. A good way to build a culture in which risk taking is accepted is to let your team know that it's OK to fail.

So, let's get that out of the way: it's OK to fail. In fact, we like to think of failure as a way of learning a lot really quickly (providing that you're not repeatedly failing at the same thing). In addition, it's important to see failure as an opportunity to learn and not to point fingers or assign blame. Failing fast is good, because there's not a lot at stake. Failing slowly can also teach a valuable lesson, but it is more painful because more is at risk and more can be lost (usually engineering time). Failing in a manner that affects your customers is probably the least desirable failure that we encounter, but it's also one in which we have the greatest amount of structure in place to learn from failures. As mentioned earlier, every time there is a major production failure at Google, we perform a postmortem. This procedure is a way to document the events that led to the actual failure and to develop a series of steps that will prevent it from happening in the future. This is neither an opportunity to point fingers, nor is it intended to introduce unnecessary bureaucratic checks; rather, the goal is to strongly focus on the core of the problem and fix it once and for all. It's very difficult, but quite effective (and cathartic).

Individual successes and failures are a bit different. It's one thing to laud individual successes, but looking to assign individual blame in the case of failure is a great way to divide a team and discourage risk taking across the board. It's alright to fail, but fail as a team and learn from your failures. If an individual succeeds, praise them in front of the team. If an individual fails, give constructive criticism in private.[5] Whatever the case, take advantage of the opportunity and apply a liberal helping of humility, respect, and trust to help your team to learn from its failures.

Antipatterns

Before we go over a litany of "design patterns" for successful TLs and engineering managers, we're going to review a collection of the patterns that you *don't* want to follow if you want to be a successful manager. We've observed these destructive patterns in a handful of bad managers that we've encountered in our careers, and in more than a few cases, ourselves.

Antipattern: Hire Pushovers

If you're a manager and you're feeling insecure in your role (for whatever reason), one way to make sure no one questions your authority or threatens your job is to hire people you can push around. You can achieve this by hiring people who aren't as smart or ambitious as you are, or just people who are more insecure than you. Even though this will cement your position as the team leader and decision maker, it will mean a lot more work for you. Your team won't be able to make a move without you leading them like dogs on a leash. If you build a team of pushovers, you probably can't take a vacation; the moment you leave the room, productivity comes to a screeching halt. But surely this is a small price to pay for feeling secure in your job, right?

Instead, you should strive to hire people who are smarter than you and can replace you. This can be difficult because these very same people will challenge you on a regular basis (in addition to letting you know when you make a mistake). These very same people will also consistently impress you and make great things happen. They'll be able to direct themselves to a much greater extent, and some will be eager to lead the team, as well. You shouldn't see this as an attempt to usurp your power; instead, look at it as an opportunity for you to lead an additional team, investigate new opportunities, or even take a vacation without worrying about checking in on the team every day to make sure it's getting its work done. It's also a great chance to learn and grow—it's a lot easier to expand your expertise when surrounded by people who are smarter than you.

Antipattern: Ignore Low Performers

Early in my career as a manager at Google, the time came for me to hand out bonus letters to my team, and I grinned as I told my manager, "I love being a manager!" Without missing a beat, my manager, a long-time industry veteran, replied, "Sometimes you get to be the tooth fairy, other times you have to be the dentist."

It's never any fun to pull teeth. We've seen team leaders do all the right things to build incredibly strong teams, only to have these teams fail to excel

(and eventually fall apart) because of just one or two low performers. We understand that the human aspect is the most challenging part of writing software, but the most difficult part of dealing with humans is handling someone who isn't meeting expectations. Sometimes, people miss expectations because they're not working long enough or hard enough, but the most difficult cases are when someone just isn't capable of doing their job no matter how long or hard they work.

Google's Site Reliability Engineering (SRE) team has a motto: "Hope is not a strategy." And nowhere is hope more overused as a strategy than in dealing with a low performer. Most team leaders grit their teeth, avert their eyes, and just *hope* that the low performer either magically improves or just goes away. Yet it is extremely rare that this person does either.

While the leader is hoping and the low performer isn't improving (or leaving), high performers on the team waste valuable time pulling the low performer along and team morale leaks away into the ether. You can be sure that the team knows the low performer is there even if you're ignoring them—in fact, the team is *acutely* aware of who the low performers are, because they have to carry them.

Ignoring low performers is not only a way to keep new high performers from joining your team, but it's also a way to encourage existing high performers to leave. You eventually wind up with an entire team of low performers because they're the only ones who can't leave of their own volition. Lastly, you aren't even doing the low performer any favors by keeping them on the team; often, someone who wouldn't do well on your team could actually have plenty of impact somewhere else.

The benefit of dealing with a low performer as quickly as possible is that you can put yourself in the position of helping them up or out. If you immediately deal with a low performer, you'll often find that they merely need some encouragement or direction to slip into a higher state of productivity. If you wait too long to deal with a low performer, their relationship with the team is going to be so sour and you're going to be so frustrated that you're not going to be able to help them.

How do you effectively coach a low performer? The best analogy is to imagine that you're helping a limping person learn to walk again, then jog, then run alongside the rest of the team. It almost always requires temporary micromanagement, but still a whole lot of humility, respect, and trust—particularly respect. Set up a specific time frame (say, two months), and some very specific goals you expect him to achieve in that period. Make the goals

small, incremental, and measurable so that there's an opportunity for lots of small successes. Meet with the team member every week to check on progress, and be sure you set really explicit expectations around each upcoming milestone so that it's easy to measure success or failure. If the low performer can't keep up, it will become quite obvious to both of you early in the process. At this point, the person will often acknowledge that things aren't going well and decide to quit; in other cases, determination will kick in and they'll "up their game" to meet expectations. Either way, by working directly with the low performer, you're catalyzing important and necessary changes.

Antipattern: Ignore Human Issues

A manager has two major areas of focus for their team: the social and the technical. It's rather common for managers to be stronger in the technical side at Google, and because most managers are promoted from a technical job (for which the primary goal of their job was to solve technical problems), they can tend to ignore human issues. It's tempting to focus all of your energy on the technical side of your team because, as an individual contributor, you spend the vast majority of your time solving technical problems. When you were a student, your classes were all about learning the technical ins and outs of your work. Now that you're a manager, however, you ignore the human element of your team at your own peril.

Let's begin with an example of a leader ignoring the human element in his team. Years ago, Jake had his first child. Jake and Katie had worked together for years, both remotely and in the same office, so in the weeks following the arrival of the new baby, Jake worked from home. This worked out great for the couple, and Katie was totally fine with it because she was already used to working remotely with Jake. They were their usual productive selves until their manager, Pablo (who worked in a different office), found out that Jake was working from home for most of the week. Pablo was upset that Jake wasn't going into the office to work with Katie, despite the fact that Jake was just as productive as always and that Katie was fine with the situation. Jake attempted to explain to Pablo that he was just as productive as he would be if he came into the office and that it was much easier on him and his wife for him to mostly work from home for a few weeks. Pablo's response: "Dude, people have kids all the time. You need to go into the office." Needless to say, Jake (normally a mild-mannered engineer) was enraged and lost a lot of respect for Pablo.

There are numerous ways in which Pablo could have handled this differently: he could have showed some understanding that Jake wanted to spend more time at home for his wife and, if his productivity and team weren't being affected, just let him continue to do so for a while. He could have negotiated that Jake go into the office for one or two days a week until things settled down. Regardless of the end result, a little bit of empathy would have gone a long way toward keeping Jake happy in this situation.

Antipattern: Be Everyone's Friend

The first foray that most people have into leadership of any sort is when they become the manager or TL of a team of which they were formerly members. Many leads don't want to lose the friendships they've cultivated with their teams, so they will sometimes work extra hard to maintain friendships with their team members after becoming a team lead. This can be a recipe for disaster and for a lot of broken friendships. Don't confuse friendship with leading with a soft touch: when you hold power over someone's career, they might feel pressure to artificially reciprocate gestures of friendship.

Remember that you can lead a team and build consensus without being a close friend of your team (or a monumental hard-ass). Likewise, you can be a tough leader without tossing your existing friendships to the wind. We've found that having lunch with your team can be an effective way to stay socially connected to them without making them uncomfortable—this gives you a chance to have informal conversations outside the normal work environment.

Sometimes, it can be tricky to move into a management role over someone who has been a good friend and a peer. If the friend who is being managed is not self-managing and is not a hard worker, it can be stressful for everyone. We recommend that you avoid getting into this situation whenever possible, but if you can't, pay extra attention to your relationship with those folks.

Antipattern: Compromise the Hiring Bar

Steve Jobs once said: "A people hire other A people; B people hire C people." It's incredibly easy to fall victim to this adage, and even more so when you're trying to hire quickly. A common approach I've seen outside of Google is that a team needs to hire five engineers, so it sifts through a pile of applications, interviews 40 or 50 people, and picks the best five candidates regardless of whether they meet the hiring bar.

This is one of the fastest ways to build a mediocre team.

The cost of finding the appropriate person—whether by paying recruiters, paying advertising, or pounding the pavement for references—pales in comparison to the cost of dealing with an employee who you never should have hired in the first place. This "cost" manifests itself in lost team productivity, team stress, time spent managing the employee up or out, and the paperwork and stress involved in firing the employee. That's assuming, of course, that you try to avoid the monumental cost of just leaving them on the team. If you're managing a team for which you don't have a say over hiring and you're unhappy with the hires being made for your team, you need to fight tooth and nail for higher-quality engineers. If you're still handed substandard engineers, maybe it's time to look for another job. Without the raw materials for a great team, you're doomed.

Antipattern: Treat Your Team Like Children

The best way to show your team that you don't trust it is to treat team members like kids—people tend to act the way you treat them, so if you treat them like children or prisoners, don't be surprised when that's how they behave. You can manifest this behavior by micromanaging them or simply by being disrespectful of their abilities and giving them no opportunity to be responsible for their work. If it's permanently necessary to micromanage people because you don't trust them, you have a hiring failure on your hands. Well, it's a failure unless your goal was to build a team that you can spend the rest of your life babysitting. If you hire people worthy of trust and show these people you trust them, they'll usually rise to the occasion (sticking with the basic premise, as we mentioned earlier, that you've hired good people).

The results of this level of trust go all the way to more mundane things like office and computer supplies. As another example, Google provides employees with cabinets stocked with various and sundry office supplies (e.g., pens, notebooks, and other "legacy" implements of creation) that are free to take as employees need them. The IT department runs numerous "Tech Stops" that provide self-service areas that are like a mini electronics store. These contain lots of computer accessories and doodads (power supplies, cables, mice, USB drives, etc.) that would be easy to just grab and walk off with *en masse*, but because Google employees are being entrusted to check these items out, they feel a responsibility to Do The Right Thing. Many people from typical corporations react in horror to hearing this, exclaiming that surely Google is hemorrhaging money due to people "stealing" these items. That's certainly possible, but what about the costs of having a workforce that behaves like children or that has to waste valuable

time formally requesting cheap office supplies? Surely that's more expensive than the price of a few pens and USB cables.

Positive Patterns

Now that we've covered antipatterns, let's turn to positive patterns for successful leadership and management that we've learned from our experiences at Google, from watching other successful leaders and, most of all, from our own leadership mentors. These patterns are not only those that we've had great success implementing, but the patterns that we've always respected the most in the leaders who we follow.

Lose the Ego

We talked about "losing the ego" a few chapters ago when we first examined humility, respect, and trust, but it's especially important when you're a team leader. This pattern is frequently misunderstood as encouraging people to be a doormat and let others walk all over them, but that's not the case at all. Of course, there's a fine line between being humble and letting others take advantage of you, but humility is not the same as lacking confidence. You can still have self-confidence and opinions without being an egomaniac. Big personal egos are difficult to handle on any team, especially in the team's leader. Instead, you should work to cultivate a strong collective team ego and identity.

Part of "losing the ego" is trust: you need to trust your team. That means respecting the abilities and prior accomplishments of the team members, even if they're new to your team.

If you're not micromanaging your team, you can be pretty certain the folks working in the trenches know the details of their work better than you do. This means that although you might be the one driving the team to consensus and helping to set the direction, the nuts and bolts of how to accomplish your goals are best decided by the people who are putting the product together. This gives them not only a greater sense of ownership, but also a greater sense of accountability and responsibility for the success (or failure) of their product. If you have a good team and you let it set the bar for the quality and rate of its work, it will accomplish more than by you standing over team members with a carrot and a stick.

Most people new to a leadership role feel an enormous responsibility to get everything right, to know everything, and to have all the answers. We can

assure you that you will not get everything right, nor will you have all the answers, and if you act like you do, you'll quickly lose the respect of your team. A lot of this comes down to having a basic sense of security in your role. Think back to when you were an individual contributor; you could smell insecurity a mile away. Try to appreciate inquiry: when someone questions a decision or statement you made, remember that this person is usually just trying to better understand you. If you encourage inquiry, you're much more likely to get the kind of constructive criticism that will make you a better leader of a better team. Finding people who will give you good constructive criticism is incredibly difficult, and it's even more difficult to get this kind of criticism from people who "work for you." Think about the big picture of what you're trying to accomplish as a team, and accept feedback and criticism openly; avoid the urge to be territorial.

The last part of losing the ego is a simple one, but many engineers would rather be boiled in oil than do it: apologize when you make a mistake. And we don't mean you should just sprinkle "I'm sorry" throughout your conversation like salt on popcorn—you need to sincerely mean it. You are absolutely going to make mistakes, and whether or not you admit it, your team is going to know you've made a mistake. Your team members will know regardless of whether they talk to you (and one thing is guaranteed: they *will* talk about it with one another). Apologizing doesn't cost money. People have enormous respect for leaders who apologize when they screw up, and contrary to popular belief, apologizing doesn't make you vulnerable. In fact, you'll usually gain respect from people when you apologize, because apologizing tells people that you are level headed, good at assessing situations, and—coming back to humility, respect, and trust—humble.

Be a Zen Master

As an engineer, you've likely developed an excellent sense of skepticism and cynicism, but this can be a liability when you're trying to lead a team. This is not to say that you should be naïvely optimistic at every turn, but you would do well to be less vocally skeptical while still letting your team know you're aware of the intricacies and obstacles involved in your work. Mediating your reactions and maintaining your calm is more important as you lead more people, because your team will (both unconsciously and consciously) look to you for clues on how to act and react to whatever is going on around you.

A simple way to visualize this effect is to see your company's organization chart as a chain of gears, with the individual contributor as a tiny gear with just a few teeth all the way at one end, and each successive manager above

them as another gear, ending with the CEO as the largest gear with many hundreds of teeth. This means that every time that individual's "manager gear" (with maybe a few dozen teeth) makes a single revolution, the "individual's gear" makes two or three revolutions. And the CEO can make a small movement and send the hapless employee, at the end of a chain of six or seven gears, spinning wildly! The farther you move up the chain, the faster you can set the gears below you spinning, whether or not you intend to.

Another way of thinking about this is the maxim that the leader is always on stage. This means that if you're in an overt leadership position, you are always being watched: not just when you run a meeting or give a talk, but even when you're just sitting at your desk answering emails. Your peers are watching you for subtle clues in your body language, your reactions to small talk, and your signals as you eat lunch. Do they read confidence or fear? As a leader, your job is to inspire, but inspiration is a 24/7 job. Your visible attitude about absolutely everything—no matter how trivial—is unconsciously noticed and spreads infectiously to your team.

One of the early managers at Google, Bill Coughran, a VP of engineering, had truly mastered the ability to maintain calm at all times. No matter what blew up, no matter what crazy thing happened, no matter how big the firestorm, Bill would never panic. Most of the time he'd place one arm across his chest, rest his chin in his hand, and ask questions about the problem, usually to a completely panicked engineer. This had the effect of calming them and helping them to focus on solving the problem instead of running around like a chicken with its head cut off. Some of us used to joke that if someone came in and told Bill that 19 of the company's offices had been attacked by space aliens, Bill's response would be, "Any idea why they didn't make it an even 20?"

This brings us to another Zen management trick: asking questions. When a team member asks you for advice, it's usually pretty exciting because you're finally getting the chance to fix something. That's exactly what you did for years before moving into a leadership position, so you usually go leaping into solution mode, but that is the last place you should be. The person asking for advice typically doesn't want *you* to solve their problem, but rather to help them solve it, and the easiest way to do this is to ask this person questions. This isn't to say that you should replace yourself with a Magic 8 Ball, which would be maddening and unhelpful. Instead, you can apply some humility, respect, and trust and try to help the person solve the problem on their own by trying to refine and explore the problem. This will usually lead the employee to the answer,[6] and it will be that person's answer, which leads

back to the ownership and responsibility we went over earlier in this chapter. Whether or not you have the answer, using this technique will almost always leave the employee with the impression that you did. Tricky, eh? Socrates would be proud of you.

Be a Catalyst

In chemistry, a catalyst is something that accelerates a chemical reaction, but which itself is not consumed in the reaction. One of the ways in which catalysts (e.g., enzymes) work is to bring reactants into close proximity: instead of bouncing around randomly in a solution, the reactants are much more likely to favorably interact with one another when the catalyst helps bring them together. This is a role you'll often need to play as a leader, and there are a number of ways you can go about it.

One of the most common things a team leader does is to build consensus. This might mean that you drive the process from start to finish, or you just give it a gentle push in the right direction to speed it up. Working to build team consensus is a leadership skill that is often used by unofficial leaders because it's one way you can lead without any actual authority. If you have the authority, you can direct and dictate direction, but that's less effective overall than building consensus.[7] If your team is looking to move quickly, sometimes it will voluntarily concede authority and direction to one or more team leads. Even though this might look like a dictatorship or oligarchy, when it's done voluntarily, it's a form of consensus.

Remove Roadblocks

Sometimes, your team already has consensus about what you need to do, but it hit a roadblock and became stuck. This could be a technical or organizational roadblock, but jumping in to help the team get moving again is a common leadership technique. There are some roadblocks that, although virtually impossible for your team members to get past, will be easy for you to handle, and helping your team to understand that you're glad (and able) to help out with these roadblocks is valuable.

One time, a team spent several weeks trying to work past an obstacle with Google's legal department. When the team finally reached its collective wits' end and went to its manager with the problem, the manager had it solved in less than two hours simply because he knew the right person to contact to discuss the matter. Another time, a team needed some server resources and just couldn't get them allocated. Fortunately, the team's manager was in

communication with other teams across the company and managed to get the team exactly what it needed that very afternoon. Yet another time, one of the engineers was having trouble with an arcane bit of Java code. Although the team's manager was not a Java expert, she was able to connect the engineer to another engineer who knew exactly what the problem was. You don't need to know all the answers to help remove roadblocks, but it usually helps to know the people who do. In many cases, knowing the right person is more valuable than knowing the right answer.

Be a Teacher and a Mentor

One of the most difficult things to do as a TL is to watch a more junior team member spend three hours working on something that you know you can knock out in 20 minutes. Teaching people and giving them a chance to learn on their own can be incredibly difficult at first, but it's a vital component of effective leadership. This is especially important for new hires who, in addition to learning your team's technology and codebase, are learning your team's culture and the appropriate level of responsibility to assume. A good mentor must balance the trade-offs of a mentee's time learning versus their time contributing to their product as part of an effective effort to scale the team as it grows.

Much like the role of manager, most people don't apply for the role of mentor—they usually become one when a leader is looking for someone to mentor a new team member. It doesn't take a lot of formal education or preparation to be a mentor. Primarily, you need three things: experience with your team's processes and systems, the ability to explain things to someone else, and the ability to gauge how much help your mentee needs. The last thing is probably the most important—giving your mentee enough information is what you're supposed to be doing, but if you overexplain things or ramble on endlessly, your mentee will probably tune you out rather than politely tell you they got it.

Set Clear Goals

This is one of those patterns that, as obvious as it sounds, is solidly ignored by an enormous number of leaders. If you're going to get your team moving rapidly in one direction, you need to make sure that every team member understands and agrees on what the direction is. Imagine your product is a big truck (and not a series of tubes). Each team member has in their hand a rope tied to the front of the truck, and as they work on the product, they'll pull the truck in their own direction. If your intention is to pull the truck (or

product) northbound as quickly as possible, you can't have team members pulling every which way—you want them all pulling the truck north. If you're going to have clear goals, you need to set clear priorities and help your team decide how it should make trade-offs when the time comes.

The easiest way to set a clear goal and get your team pulling the product in the same direction is to create a concise mission statement for the team. After you've helped the team define its direction and goals, you can step back and give it more autonomy, periodically checking in to make sure everyone is still on the right track. This not only frees up your time to handle other leadership tasks, it also drastically increases the efficiency of your team. Teams can (and do) succeed without clear goals, but they typically waste a great deal of energy as each team member pulls the product in a slightly different direction. This frustrates you, slows progress for the team, and forces you to use more and more of your own energy to correct the course.

Be Honest

This doesn't mean that we're assuming you are lying to your team, but it merits a mention because you'll inevitably find yourself in a position in which you can't tell your team something or, even worse, you need to tell everyone something they don't want to hear. One manager we know tells new team members, "I won't lie to you, but I will tell you when I can't tell you something or if I just don't know."

If a team member approaches you about something you can't share, it's OK to just tell them you know the answer but are not at liberty to say anything. Even more common is when a team member asks you something you don't know the answer to: you can tell that person you don't know. This is another one of those things that seems blindingly obvious when you read it, but many people in a manager role feel that if they don't know the answer to something, it proves that they're weak or out of touch. In reality, the only thing it proves is that they're human.

Giving hard feedback is…well, hard. The first time you need to tell one of your reports that they made a mistake or didn't do their job as well as expected can be incredibly stressful. Most management texts advise that you use the "compliment sandwich" to soften the blow when delivering hard feedback. A compliment sandwich looks something like this:

You're a solid member of the team and one of our smartest engineers. That being said, your code is convoluted and almost impossible for anyone else on

the team to understand. But you've got great potential and a wicked cool T-shirt collection.

Sure, this softens the blow, but with this sort of beating around the bush most people will walk out of this meeting thinking only, "Sweet! I've got cool T-shirts!" We *strongly* advise against using the compliment sandwich, not because we think you should be unnecessarily cruel or harsh, but because most people won't hear the critical message, which is that something needs to change. It's possible to employ respect here: be kind and empathetic when delivering constructive criticism without resorting to the compliment sandwich. In fact, kindness and empathy are critical if you want the recipient to hear the criticism and not immediately go on the defensive.

Years ago, a colleague picked up a team member, Tim, from another manager who insisted that Tim was impossible to work with. He said that Tim never responded to feedback or criticism and instead just kept doing the same things he'd been told he shouldn't do. Our colleague sat in on a few of the manager's meetings with Tim to watch the interaction between the manager and Tim, and noticed that the manager made extensive use of the compliment sandwich so as not to hurt Tim's feelings. When they brought Tim onto their team, they sat down with him and very clearly explained that Tim needed to make some changes to work more effectively with the team:

We're quite sure that you're not aware of this, but the way that you're interacting with the team is alienating and angering them, and if you want to be effective, you need to refine your communication skills and we're committed to helping you do that.

They didn't give Tim any compliments or candy-coat the issue, but just as important, they weren't mean—they just laid out the facts as they saw them based on Tim's performance with the previous team. Lo and behold, within a matter of weeks (and after a few more "refresher" meetings), Tim's performance improved dramatically. Tim just needed very clear feedback and direction.

When you're providing direct feedback or criticism, your delivery is key to making sure that your message is heard and not deflected. If you put the recipient on the defensive, they're not going to be thinking of how they can change, but rather how they can argue with you to show you that you're wrong. Our colleague Ben once managed an engineer who we'll call Dean. Dean had extremely strong opinions and would argue with the rest of the team about anything. It could be something as big as the team's mission or as small as the placement of a widget on a web page; Dean would argue with

the same conviction and vehemence either way, and he refused to let anything slide. After months of this behavior, Ben met with Dean to explain to him that he was being too combative. Now, if Ben had just said, "Dean, stop being such a jerk," you can be pretty sure Dean would have disregarded it entirely. Ben thought hard about how he could get Dean to understand how his actions were adversely affecting the team, and he came up with the following metaphor:

Every time a decision is made, it's like a train coming through town—when you jump in front of the train to stop it, you slow the train down and potentially annoy the engineer driving the train. A new train comes by every 15 minutes, and if you jump in front of every train, not only do you spend a lot of your time stopping trains, but eventually one of the engineers driving the train is going to get mad enough to run right over you. So, although it's OK to jump in front of some trains, pick and choose the ones you want to stop to make sure you're stopping only the trains that really matter.

This anecdote not only injected a bit of humor into the situation, but also made it easier for Ben and Dean to discuss the effect that Dean's "train stopping" was having on the team in addition to the energy Dean was spending on it.

Track Happiness

As a leader, one way you can make your team more productive (and less likely to leave) in the long term is to take some time to gauge their happiness. The best leaders we've worked with have all been amateur psychologists, looking in on their team members' welfare from time to time, making sure they get recognition for what they do, and trying to make certain they are happy with their work. One TLM we know makes a spreadsheet of all the grungy, thankless tasks that need to be done and makes certain these tasks are evenly spread across the team. Another TLM watches the hours his team is working and uses comp time and fun team outings to avoid burnout and exhaustion. Yet another starts one-on-one sessions with his team members by dealing with their technical issues as a way to break the ice, and then takes some time to make sure each engineer has everything they need to get their work done. After they've warmed up, he talks to the engineer for a bit about how they're enjoying the work and what they're looking forward to next.

A good simple way to track your team's happiness[8] is to ask the team member at the end of each one-on-one meeting, "What do you need?" This simple question is a great way to wrap up and make sure each team member has what they need to be productive and happy, although you might need to

carefully probe a bit to get details. If you ask this every time you have a one-on-one, you'll find that eventually your team will remember this and sometimes even come to you with a laundry list of things it needs to make everyone's job better.

The Unexpected Question

Shortly after I started at Google, I had my first meeting with then-CEO Eric Schmidt, and at the end Eric asked me, "Is there anything you need?" I had prepared a million defensive responses to difficult questions or challenges but was completely unprepared for this. So I sat there, dumbstruck and staring. You can be sure I had something ready the next time I was asked that question!

It can also be worthwhile as a leader to pay some attention to your team's happiness outside the office. Our colleague Mekka starts his one-on-ones by asking his reports to rate their happiness on a scale of 1 to 10, and oftentimes his reports will use this as a way to discuss happiness in *and* outside of the office. Be wary of assuming that people have no life outside of work—having unrealistic expectations about the amount of time people can put into their work will cause people to lose respect for you, or worse, to burn out. We're not advocating that you pry into your team members' personal lives, but being sensitive to personal situations that your team members are going through can give you a lot of insight as to why they might be more or less productive at any given time. Giving a little extra slack to a team member who is currently having a tough time at home can make them a lot more willing to put in longer hours when your team has a tight deadline to hit later.

A big part of tracking your team members' happiness is tracking their careers. If you ask a team member where they see their career in five years, most of the time you'll get a shrug and a blank look. When put on the spot, most people won't say much about this, but there are usually a few things that everyone would like to do in the next five years: be promoted, learn something new, launch something important, and work with smart people. Regardless of whether they verbalize this, most people are thinking about it. If you're going to be an effective leader, you should be thinking about how you can help make all those things happen and let your team know you're thinking about this. The most important part of this is to take these implicit goals and make them explicit so that when you're giving career advice you have a real set of metrics on which to evaluate situations and opportunities.

Tracking happiness comes down to not just monitoring careers, but also giving your team members opportunities to improve themselves, be recognized for the work they do, and have a little fun along the way.

Other Tips and Tricks

Following are other, miscellaneous tips and tricks that we at Google would recommend when you're in a leadership position:

Delegate, but get your hands dirty

When moving from an individual contributor role to a leadership role, achieving a balance is one of the most difficult things to do. Initially, you're inclined to do all of the work yourself, and after being in a leadership role for a long time, it's easy to get into the habit of doing none of the work yourself. If you're new to a leadership role, you probably need to work hard to delegate work to other engineers on your team, even if it will take them a lot longer than you to accomplish that work. Not only is this one way for you to maintain your sanity, but also it's how the rest of your team will learn. If you've been leading teams for a while or if you pick up a new team, one of the easiest ways to gain the team's respect and get up to speed on what they're doing is to get your hands dirty—usually by taking on a grungy task that no one else wants to do. You can have a résumé and a list of achievements a mile long, but nothing lets a team know how skillful and dedicated (and humble) you are like jumping in and actually doing some hard work.

Seek to replace yourself

Unless you want to keep doing the exact same job for the rest of your career, seek to replace yourself. This starts, as we mentioned earlier, with the hiring process: if you want a member of your team to replace you, you need to hire people capable of replacing you, which we usually sum up by saying you need to "hire people smarter than you." After you have team members capable of doing your job, you need to give them opportunities to take on more responsibilities or occasionally lead the team. If you do this, you'll quickly see who has the most aptitude to lead as well as who wants to lead the team. Remember that some people prefer to just be high-performing

individual contributors, and that's OK. We've always been amazed at companies that take their best engineers and—against their wishes—throw these engineers into management roles. This usually subtracts a great engineer from your team and adds a subpar manager.

Know when to make waves

You will (inevitably and frequently) have difficult situations crop up in which every cell in your body is screaming at you to do nothing about it. It might be the engineer on your team whose technical chops aren't up to par. It might be the person who jumps in front of every train. It might be the unmotivated employee who is working 30 hours a week. "Just wait a bit and it will get better," you'll tell yourself. "It will work itself out," you'll rationalize. Don't fall into this trap—these are the situations for which you need to make the biggest waves and you need to make them now. Rarely will these problems work themselves out, and the longer you wait to address them, the more they'll adversely affect the rest of the team and the more they'll keep you up at night thinking about them. By waiting, you're only delaying the inevitable and causing untold damage in the process. So act, and act quickly.

Shield your team from chaos

When you step into a leadership role, the first thing you'll usually discover is that outside your team is a world of chaos and uncertainty (or even insanity) that you never saw when you were an individual contributor. When I first became a manager back in the 1990s (before going back to being an individual contributor), I was taken aback by the sheer volume of uncertainty and organizational chaos that was happening in my company. I asked another manager what had caused this sudden rockiness in the otherwise calm company, and the other manager laughed hysterically at my naïveté: the chaos had always been present, but my previous manager had shielded me and the rest of my team from it.

Give your team air cover

Whereas it's important that you keep your team informed about what's going on "above" them in the company, it's just as important that you defend them from a lot of the uncertainty and frivolous demands that can be imposed upon you from outside your team.

Share as much information as you can with your team, but don't distract them with organizational craziness that is extremely unlikely to ever actually affect them.

Let your team know when they're doing well

Many new team leads can get so caught up in dealing with the shortcomings of their team members that they neglect to provide positive feedback often enough. Just as you let someone know when they screw up, be sure to let them know when they do well, and be sure to let them (and the rest of the team) know when they knock one out of the park.

Lastly, here's something the best leaders know and use often when they have adventurous team members who want to try new things:

it's easy to say "yes" to something that's easy to undo

If you have a team member who wants to take a day or two to try using a new tool or library[9] that could speed up your product (and you're not on a tight deadline), it's easy to say, "Sure, give it a shot." If, on the other hand, they want to do something like launch a product that you're going to have to support for the next 10 years, you'll likely want to give it a bit more thought. Really good leaders have a good sense for when something can be undone, but more things are undoable than you think (and this applies to both technical and nontechnical decisions).

People Are Like Plants

My wife is the youngest of six children, and her mother was faced with the difficult task of figuring out how to raise six very different children, each of whom needed different things. I asked my mother-in-law how she managed this (see what I did there?), and she responded that kids are like plants: some are like cacti and need little water but lots of sunshine, others are like African violets and need diffuse light and moist soil, and still others are like tomatoes and will truly excel if you give them a little fertilizer. If you have six kids and give each one the same amount of water, light, and fertilizer, they'll all get equal treatment, but the odds are good that *none* of them will get what they actually need.

And so your team members are also like plants: some need more light, and some need more water (and some need more…fertilizer). It's your job as their leader to determine who needs what and then give it to them—except instead of light, water, and fertilizer, your team needs varying amounts of motivation and direction.

To get all of your team members what they need, you need to motivate the ones who are in a rut, and provide stronger direction to those who are distracted or uncertain of what to do. Of course, there are those who are "adrift" and need both motivation and direction. So, with this combination of motivation and direction you can make your team happy and productive. And you don't want to give them too much of either—because if they don't need motivation or direction and you try giving it to them, you're just going to annoy them.

Giving direction is fairly straightforward—it requires a basic understanding of what needs to be done, some simple organizational skills, and enough coordination to break it down into manageable tasks. With these tools in hand you can provide sufficient guidance for an engineer in need of directional help. Motivation, however, is a bit more sophisticated and merits some explanation.

Intrinsic versus Extrinsic Motivation

There are two types of motivation: *extrinsic*, which originates from outside forces (such as monetary compensation), and *intrinsic*, which comes from within. In his book *Drive*,[10] Dan Pink explains that the way to make people the happiest and most productive isn't to motivate them extrinsically (e.g., throw piles of cash at them); rather, you need to work to increase their intrinsic motivation. Dan claims you can increase intrinsic motivation by giving people three things: autonomy, mastery, and purpose.[11]

A person has autonomy when they have the ability to act on their own without someone micromanaging them[12]. With autonomous employees (and Google strives to hire mostly autonomous engineers), you might give them the general direction in which they need to take the product but leave it up to them to decide how to get there. This helps with motivation not only because they have a closer relationship with the product (and likely know better than you how to build it), but also because it gives them a much greater sense of ownership of the product. The bigger their stake is in the success of the product, the greater their interest is in seeing it succeed.

Mastery in its basest form simply means that you need to give someone the opportunity to improve existing skills and learn new ones. Giving ample opportunities for mastery not only helps to motivate people, but also makes them better over time, which makes for stronger teams.[13] An employee's skills are like the blade of a knife: you can spend tens of thousands of dollars to find people with the sharpest skills for your team, but if you use that knife for years without sharpening it, you will wind up with a dull knife that is inefficient, and in some cases useless. Google gives ample opportunities for engineers to learn new things and master their craft so as to keep them sharp, efficient, and effective.

Of course, all the autonomy and mastery in the world isn't going to help motivate someone if they're doing work for no reason at all, which is why you need to give their work purpose. Many people work on products that have great significance, but they're kept at arm's length from the positive effects their products might have on their company, their customers, or even the world. Even for cases in which the product might have a much smaller impact, you can motivate your team by seeking the reason for their efforts and making this reason clear to them. If you can help them to see this purpose in their work, you'll see a tremendous increase in their motivation and productivity.[14] One manager we know keeps a close eye on the email feedback that Google gets for its product (one of the "smaller-impact" products), and whenever she sees a message from a customer talking about how the company's product has helped the customer personally or helped the customer's business, she immediately forwards it to the engineering team. This not only motivates the team, but also frequently inspires team members to think about ways in which they can make their product even better.

Conclusion

Leading a team is a different task than that of being a software engineer. As a result, good software engineers do not always make good managers, and that's OK—effective organizations allow productive career paths to both individual contributors and people managers. Though Google has found that software engineering experience itself is invaluable for managers, the most important skills an effective manager brings to the table are social ones. Good managers enable their engineering teams by helping them work well, keeping them focused on proper goals, and insulating them from problems outside the group, all the while following the three pillars of humility, trust, and respect.

TL;DRs

- Don't "manage" in the traditional sense; focus on leadership, influence, and serving your team.
- Delegate where possible, don't DIY (Do It Yourself).
- Pay particular attention to the focus, direction, and velocity of your team.

1 Another difference that takes getting used to is that the things we do as managers typically pay off over a longer timeline.

2 Yet another reason companies shouldn't force people into management as part of a career path: if an engineer is able to write reams of great code and has no desire at all to manage people or lead a team, by forcing them into a management or TL role you're losing a great engineer and gaining a crappy manager. This is not only a bad idea, but it's actively harmful.

3 For more fascinating information on optimizing the movements of factory workers, read up on Scientific Management or Taylorism, especially its effects on worker morale.

4 If you have kids, the odds are good that you can remember with startling clarity the first time you said something to your child that made you stop and exclaim (perhaps even aloud), "Holy crap, I've become my mother."

5 Public criticism of an individual is not only ineffective (it puts people on the defense), but rarely necessary, and most often is just mean or cruel. You can be sure the rest of the team already knows when an individual has failed, so there's no need to rub it in.

6 See also "Rubber duck debugging."

7 Attempting to achieve 100% consensus can also be harmful. You need to be able to decide to proceed even if not everyone is on the same page or there is still some uncertainty.

8 Google also runs an annual employee survey called "Googlegeist" that rates employee happiness across many dimensions. This provides good feedback but isn't what we would call "simple."

9 To gain a better understanding of just how "undoable" technical changes can be, see Chapter 22.

10 See Dan's fantastic TED talk on this subject.

11 This assumes that the people in question are being paid well enough that income is not a source of stress.

12 This assumes that you have people on your team who don't need micromanagement.

13 Of course, it also means they're more valuable and marketable employees, so it's easier for them to pick up and leave you if they're not enjoying their work. See the pattern in "Track Happiness."

14 *http://bit.ly/task_significance*