Every great hire makes your life better and easier

HOW TO RECRUIT & HIRE GREAT SOFTWARE ENGINEERS

Building a Crack Development Team

Patrick McCuller

Apress®

For your convenience Apress has placed some of the front matter material after the index. Please use the Bookmarks and Contents at a Glance links to access them.

Contents

CHAPTER

1

Introduction

This chapter describes the audience and scope of this book and suggests how you can use it to recruit the best software engineers available. It explains the central themes: hiring well is a competitive advantage, treat candidates as well as you treat customers, and take an engineering approach to the recruiting and hiring process. This chapter also provides a troubleshooting table to identify the appropriate chapters for answers to the most common and easily articulated questions.

Who Should Read This Book

This book is intended for technical managers who need to hire software engineers to build core software applications. Technical managers at all levels of hiring experience will benefit from this book—from absolute beginners looking for a place to start to veterans looking for ways to optimize the hiring process. That includes software development managers, directors, chief technology officers (CTOs), and entrepreneurs.

This book is not meant to be a deep analysis of the realm of recruiting. Some topics, such as sourcing candidates, are treated lightly, as hiring managers are less likely to need to drive that process personally. You will, however, learn enough about sourcing to work with and help sourcers and to pinch-hit in that role.

The topics addressed in depth are the most useful to hiring managers, addressing critical points and issues with detail. That includes optimizing the overall process, evaluating résumés, conducting interviews, asking interview questions and interpreting answers, and maximizing the use of allies and partners, such as professional recruiters.

The software engineers who are the subjects of the hiring process described in this book work under many titles:

- Software Engineer
- Software Design Engineer
- Software Development Engineer
- Software Development Engineer in Test
- Principal Engineer
- Programmer
- Lead Engineer
- Java Developer
- UI Developer
- .NET Developer
- Systems Analyst

To a lesser degree, this book will also be helpful when hiring people with the following job titles:

- Operations Engineer
- Support Engineer
- Software Manager
- Information Architect
- General IT staff

How to Use This Book If You're Pressed for Time

The chapters of this book are modular, in the sense that you may read a given chapter in isolation from preceding chapters. After reading this introduction, you may jump to the parts that are specifically relevant to your current recruiting needs. Here's a quick troubleshooting guide.

Not sure what kind of person to hire?	Chapter 1
Process is slow or unwieldy?	Chapter 2
Not finding enough candidates?	Chapter 3
Résumés don't help distinguish good candidates from bad candidates?	Chapter 4
Interviews aren't going well, or not hiring at all?	Chapter 5
Interview questions are mysterious?	Chapter 6
Hiring decisions are difficult or random?	Chapter 7
Candidates don't accept offers?	Chapter 8
New hires don't thrive?	Chapter 9

Content Overview

There are several professions dedicated to aspects of recruiting and entire sciences dedicated to measuring human capability. This book is not a replacement for or even an introduction to these professions and sciences. It is a compendium of the practical knowledge, heuristics, and tips that I have found and observed to be critically useful for managers hiring software engineers.

Chapters 2, 3, and 4 are concerned with defining what sort of engineers you want, evaluating and optimizing the overall process of hiring them, and finding candidates, respectively.

Chapters 5, 6, and 7 drill down into the interview process: reading résumés, running interviews, and creating and asking technical questions.

Chapters 8, 9, and 10 discuss hiring decisions, making offers, and getting newly hired engineers off to a great start.

Legal Disclaimer

Hiring is closely regulated by federal and state law. Although the information I present here is, to the best of my knowledge, both practical and legal in my jurisdiction at the time of writing, I am not a lawyer and I proffer no legal advice in this book. Always consult your company's human resources (HR) department and counsel before making changes to your hiring process.

Analytic versus Intuitive Styles

Everyone has a different style of approach to solving problems. Some rely on intuition; others rely on analysis. Any person usually uses a combination of these methods with one predominating.

The stereotype for engineers and engineering managers is that of a highly analytical person who uses careful reasoning, charts, logic, and mathematics to make decisions. The truth is very different: most people are not analytical most of the time.[1] In my experience, engineers analyze only slightly more than the average person does. That modicum of analysis is usually sufficient, but it's a mistake to assume that there is a careful framework behind each of an engineer's decisions.

It is possible to be quite successful in any number of fields going by intuition and rough estimation. I present analytical tools in this work—such as maintaining and analyzing detailed records of candidates, interviews, and interviewers—but I do not condemn or deprecate decision making by other means. As a professional, you should rely on the tools that you know work. The existence of an efficient tool is not sufficient reason to compel you to learn it and use it. All tools require training and investment, and the more they resemble tools you're familiar with, the easier they will be to master. If you're an intuitive person, some of the tools and methods in this work may seem like overkill to you.

That's all right. This is not an all-or-nothing system. Pick out the parts that are sensible and feasible for you and your situation, and disregard the rest or stash them away for future use. Where there are tools and methods that work most effectively in concert together, I point that out. Adapt the ideas and make them your own. If you can make sensible judgments without a calculus, go right ahead. If you find that the analytical results don't match your experience, needs, or sensibility, do the right thing for you and your business.

As I tell my employees: never suspend your judgment (especially when I ask you to).

The Competitive Advantage

The day-to-day activity of the employees of a company at all levels—from broad strategy set by executives in the C-suite to the commonplace choices and prioritizations made by interns in the mailroom—drives success in the short and long run. Each person contributes to the whole by performing or failing to perform every day.

Productivity compounds. Improvement, optimization, eliminating waste and unnecessary tasks, deftly creating a brilliant customer experience with small and easy touches—all the natural and continuous *kaizen* that occurs by the actions of talented, skilled, and motivated employees—will drive (or in its absence, fail to drive) growth and success.

---

[1] See, for example, Daniel Kahneman, *Thinking, Fast and Slow* (New York: FSG, 2011).

Employees are the lifeblood of the organization—your organization—and your success depends on their success. You set up them up for it with tools, comfortable environments, encouragement, guidance, strong coworkers, and appropriate and strategically important goals, but the raw material of talent and prehoned skills you start with determines *how* they use this and *whether* they achieve today's and tomorrow's goals. As a result, a discrepancy in employee capability between competing companies will drive a widening gap in their productivity and innovation levels, which in turn results in a gap in serving customers and overall competitiveness.

Over the years since the formation of the modern software industry, there have appeared tools such as career websites, knowledgeable headhunters, boutique recruiting firms, résumé repositories, recruiting coordination systems, and some specialized HR roles. General momentum in this area has improved the situation across the industry, and companies willing to invest in tools, process, staffing, and training have benefited even more.

In the big picture of business optimization, the science of effective technical hiring has barely changed. It has not received a fraction of the attention and effort that has gone into logistics, construction process, or development tools, such as languages, compilers, and integrated development environments.

I suspect this is due to a general perception that hiring is driven by luck or intuition, and that the managers who tackle these problems are generally uncomfortable or unfamiliar with the process and working for or with recruiters who are not especially technically minded.

However we got here, there are opportunities to do much better than average. Bring in better talent and more skilled employees and you have built or shored up the foundations of a great and successful business.

A small difference in hiring effectiveness will amplify over time into a substantial competitive advantage. You don't need to have the best recruiting and hiring method in existence, but any improvement you make and any movement above average will benefit you considerably.

Central Ideas

While engineering my own recruiting processes, I found three key ideas that reliably improved each step and decision. These themes, elaborated here, are taking an engineering approach, relying on evidence, and treating candidates as customers.

An Engineering Approach

Recruiting is a process with inputs and outputs and actions in between, so it's within the realm of engineering, and we can treat it as an engineerable subject.

This book is not framed as "Here's my way, which is the right way"—but is instead an examination of the process and its components, highlighting the role of thoughtful, intentional choices. It's your process because you own the outcome. That outcome is extremely important to your success, so it deserves your attention and the application of your hard-won skills, energy, resources, and creativity.

You may have inherited a process (or find yourself inside one), but that process is not everything it can be or must be to drive your success. Instead, think of that process as a prototype and a list of parts. Look at it like an engineer would: figure out how it works, how it doesn't work, and what makes it run quickly and slowly. Take care with what you put into it and how it behaves, and examine the output regularly to make sure it's running the way you want and need it to run.

Every section in this book describes recurrent realities and discusses options for dealing with them and succeeding. It's not a "this-is-my-method-follow-it-and-you-will-be-successful" book because that's not realistic. What I do varies with time and circumstance, and your time and circumstances vary from mine. It would be impertinent to insist on a particular hiring process.

Instead, I am lending you my perspective on the nature and purpose of the general hiring process for software engineers and many of the options you have available while building and tinkering with your own hiring machine. It's an approach that says, "Here are the kinds of parts that fit into this slot, and what's worked for me best has been XYZ—but keep in mind that I don't necessarily know all the parts that fit into your particular slot."

The keys to great engineering are meticulousness and creativity. Attention to detail, and returning your attention to it over time, will drive your ability to find what needs to change. Designing an optimal new process will take all your creativity, and the very best process you can craft will take you past what's in this book and beyond what anyone currently knows how to make.

Evidence-Based Hiring

Effective decisions require data that represent reality. Much of the information we need to make great hiring decisions is hidden in noise: inaccurate résumés, irrelevant personal data, ambiguous answers to interview questions, and so on. The purpose of interviewing is to identify the useful information by separating signal from noise, so we must be diligent and thorough in rooting out the truth.

Paired with the need for good data is a perennial need for effective decision making. All too frequently we lead with intuition and back it up with whatever evidence is at hand. We have many biases that we don't normally detect or even know exist, and our colleagues in the recruiting process have them, too.

Humans have built-in cognitive defects that we can adjust for if we are vigilant and purposeful, understanding and working with our limitations. We are unconsciously prone to anchoring, confirmation bias, framing effects, and all sorts of pitfalls described in chapters 4 through 7.

Candidates as Customers

Long-term success is usually the result of consistently applied strategy and principles. This book takes an engineering approach to the strategy and tactics of hiring; making a sound decision also involves applying behavioral principles rooted in human feelings. The principle I have had the most success with is that candidates are customers.

Thinking of candidates as customers caused a shift in the way I treated them and my overall approach to fine-tuning the recruiting process. Teaching my teams to think of candidates as customers not only elevated the candidates' experience when interviewing with us, it also increased my teams' empathy with the candidates. The interviewers thought more deeply about their own actions and interpreted candidates' behaviors more in the context of potentially working together—and less against an abstract ideal. Allies in the recruiting team appreciated how much easier it was to work with happier candidates, and negotiations became more likely to succeed. It also avoided losing or alienating real customers, and every customer counts.

Employees have tremendous investment in their work, from acquiring a job to putting in daily effort to move products forward, so they typically take job hunting quite seriously. You may remember doing so yourself!

Software engineers know that there's a lot riding on landing the right job. They need to secure suitable compensation, they need an environment to grow their skills and career, they need a job that maintains their interest and stimulates their active and powerful minds, and they need a bit of dependable stability. Too much stress can hurt them; too little can demotivate. Engineers can lose their jobs at any moment through a company's caprice, so they must have a basic sense of trust that the employer is stable, reasonable, and humane.

You can establish that trust with your own behavior. The hiring process has a large number of actions that a candidate can see, so they are exposed to a lot of what your company does—more specifically, what you do—and will draw conclusions about your character.

All people want to work with and for people who respect and care about others—the fundamental hallmarks of good customer service. Treating all candidates with respect and (affordable) deference makes for a smooth and effective process all around. Your players—interviewers, recruiters, and so on—will know the right thing to do most of the time, whether they are trying to follow a well-established procedure book or winging it.

Last, hiring is a human process. You find, evaluate, hire, or pass on folks just like yourself. This critical idea boils down to a simple prescription: *treat candidates as human beings.*

CHAPTER

2

Talent Management

This chapter describes engineer work styles, planning for team and product evolution, and the importance of hiring only the best engineers.

Team Planning

As a development manager, you have goals and deadlines for products and particular customer needs, and that is the day-to-day part of your business. You also have innovation and strategic maneuvering, which drive tomorrow's goals and deadlines. Capable leadership will identify and guide the organization toward shifting goals over time, perhaps rapidly, so your teams should be prepared to meet today's and tomorrow's needs.

Specialists and Generalists

Two categories of tools are required to build software products. First are the tools at hand: the methods and construction tools you use to build software and solutions to meet current goals. I call these *immediate tools*. Second, meta-tools allow you to identify, learn, and when necessary construct the new immediate tools you'll need to solve the next set of goals. I call these *capability tools*.

It is easy to find engineers who have experience and expertise with a limited domain of immediate tools, relatively speaking. Hiring exclusively these engineers may give you the ability to deliver on today's goals, but if these people are not also experts with capability tools, then your products may stagnate

over time, and your team may be unable to adapt to meet new goals in a rapidly shifting industry.

Many software engineering teams are expected to be long-lived, creating and redefining a product or product class over time, or moving from one product or goal set to the next and doing it again. Because of these expectations, there is a great need to avoid stagnation. Unfortunately, it is difficult to see at hiring time what the long-term effects of hiring just immediate tools experts will be, as the difference between the team's capability and the team's needs will start small and grow over time, usually slowly enough that the cause of the attenuating effectiveness becomes lost or hard to spot in a sea of other difficulties.

Whenever possible, it is best to hire engineers who are excellent with capability tools, in preference to expertise with immediate tools. Capability tools include the ability to learn, master, and teach:

- New construction methods
- New programming languages
- New techniques
- New platforms
- New software domains, such as embedded systems or cloud services
- New knowledge domains, such as statistical analysis or 3D simulations
- Recognizing the need to innovate
- Innovating and building new tools
- Collaborating with experts in disparate roles and domains

Of course there are and should be specialists, by which I mean people who have put substantial time into mastering a relatively narrow domain of software engineering, such as database design, compiler architecture, or machine learning, as well as people with who are also specialists in nonsoftware domains such as statistics, molecular biology, and the like—whatever applies to your team, products, and customers. Specialist experts seem to be all too rare when you actually need them.

At first, specialists may be rapidly productive on your team, with their command of the particular immediate tools you use right now. However, generalists, with command of general-purpose capability tools, can rapidly acquire new skills, adapt to evolving technology, and build new technologies to keep your products on the cutting edge.

Your engineers must be able to recognize when they need to find new tools, and then use those tools, and repeat as necessary to deliver to every goal and continuously improve your team's overall capability. You must find and hire such engineers.

Capability: Set Your Sights High

In this chapter and elsewhere in this book, I strongly advocate only targeting and hiring the top engineers in the market. You don't need justification to hire top performers, but you might believe you need justification to hire *only* top performers. There are many possible objections to focusing on the top. Here are a few, with my retorts:

- **"I Must Hire Quickly, So Hiring High Is a Luxury."**

  I reply that hiring low would be a luxury—as in, it is something you don't really need. Low-capability engineers will consume your time and energy and you will get less done, maybe a lot less.

- **"I Don't Have the Budget to Hire the Best."**

  I reply that the best cost only marginally more but create substantially more value. The math is simple; what you don't have is the budget to hire low.

- **"Who Will Employ the Bottom 90 Percent?"**

  I reply that someone less clever than you are will hire them.

. . . and Not Low

You may be tempted or even instructed to hire the first basically competent developers who show up for interviews. This would be an effective strategy only if the great majority of developers were excellent developers. There would still be a capability curve—Bell, Gaussian, Poisson, and so on—but the lower quartile would still be competent, be effective, and help drive your business forward. Today that's far from the case, and the simple reality is that at this time the majority of developers are not at all competent and will actually hurt your business.

Don't hire anyone but the best. You are too likely to accidentally hire below the best anyway, due to a minimal, unavoidable level of error and uncertainly in the hiring process; aim low and you could hire abysmally. Just as a high

performer can dominate the output of an otherwise mediocre group of workers, a low performer can sink it. The cost of dealing with the associated problems, the messed-up projects and missed deadlines, eventually firing the low performer, and then replacing him with *another* round of hiring may be such a net negative that you would have been better off not hiring anyone at all.

At times I have been instructed to lower my hiring standards, and for all the listed reasons I refused. That tactic might not be appropriate for your situation, but if you can even hire well *surreptitiously* you will prevent a lot of personal frustration and help your organization, maybe more than it deserves.

Another thing to keep in mind, even for your own career, is that organizations that hire low tend to deteriorate over time. It's tempting and straightforward to think that because they hired *you* they take hiring seriously and only go after the best. But everyone can make mistakes and win by accident.

Absolute Performance

There is a common understanding in the industry that the best software developers are 10:1 as productive as average ones. Sometimes it's said to be 16:1 or more, and it's held occasionally to be the difference between the best and the worst or between the best and the average. That's an impressive set of numbers, but there is good reason to doubt those figures, as the original studies behind them are doubtful in origin and method.

In his self-published book *The Leprechauns of Software Engineering,* Laurent Bossavit describes his efforts to track down and verify the ultimate, fundamental sources of these commonly held axioms. It's not a pretty story, and the ultimate conclusion is that there is no firm basis for believing those ratios.

There is undeniably a substantial productivity difference between engineers in the market, and it is more than a 10% or 50% difference. I think it's much more, but I have no quantitative characterization. I have had the experience of meeting a superstar engineer, who outperformed every other engineer I knew by quite a margin in any dimension you might care to name. Much later, I met another superstar who was as far ahead of the first as that star was ahead of others. And then another. These are exactly the sort of people you want to find (before I do).

Net Capabilities: Using a Talent Portfolio

Every engineer has strengths and weaknesses, but across an entire team you need a certain set of strengths, as well as particular skills and knowledge at the moment and for the foreseeable future. One of the tools I use for recording and planning across a team is building a portfolio.

A portfolio allows you to easily see who and what you have now, for review or reminder or even for eventually briefing a successor. (You should always keep in mind that you will have a successor one day and arm that person for success with a rock-solid team and lots of data they would otherwise have to scrape together over a long time.)

In a portfolio you can keep records of former team members as well, so you can contact them if you need to, though realistically this is not always a possibility. You can collect the job descriptions you've written and filled over time.

To build one, keep résumés on file as you hire and ask for résumés from all of your engineers. Ask your team to make periodic updates to their résumés, perhaps every quarter or so. You might worry that this will bring their attention to finding another job, because after all, that's when people tend to think about and work on their résumés! But if they are happy, relax—this won't make them find another job. Actually, it can refocus them on their current and ongoing success with your team and on making sure they spend time and energy advancing their careers—that is, improving their capabilities and pushing *your* technology and your products forward.

If your organization uses a "capabilities," "competencies," "leadership principles," or another sort of system for articulating desirable traits and behavior, the portfolio helps you track how well your team as a whole fulfills the capabilities, and you can use this information to characterize the strengths you must find in new hires. If your organization doesn't have some sort of behavior evaluation framework, now is a good time to adopt one, if only for your own purposes.

A portfolio is also useful for skill distribution measurement. Teams are vulnerable to member loss, creating a gap in understanding the product or projects and how to work with them. Too few team members with any given knowledge or skill can create a project bottleneck as well. Avoiding those problems is an important aspect of risk management, so a tool that organizes available information on what people know and can do will aid you in recovering from employee loss and lend your team flexibility in task assignments.

Natural Team Life Cycle and Personality Types

Some people like to start new things, and others like to tweak and optimize existing things. These are the general preferences engineers have, too. Individual preferences vary over time, but each engineer tends to have one of two stable core personalities. "Startup engineers" frequently remain startup engineers for a long time: I call them *Creators*. Long-haul systems engineers similarly stick to their pattern: I call them *Optimizers*. Both are critical, but the Creator set is more critical at the outset of a product, and the Optimizer set is more useful toward its sunset.

As you kick off a product, you need a heavy slate of Creator engineers to get things moving. They bring a lot of energy to making new software. This phase frequently needs fast motion and rapid iteration, and many important considerations (classically, I have found, monitors and alarms) are pushed out to the future, rather than built in from the beginning. In such a phase, a Creator will say that although these are important and useful, they don't get code in front of customers, and no one knows what the final product should be exactly, so the team is better off building first and asking questions later.

Whether this is really the right approach is debatable, but it summarizes a common attitude. When the product has been shipped to customers and at least some of the team's energy becomes devoted to keeping the product alive for them—customer support, bug fixes, operations, and so on—then the balance of need shifts to the Optimizers. They have the patience and desire to "do it right": systematically fix bugs, lower operational or support costs, refactor for scalability, and so on. They tend to thrive when there is already a product in place.

Mixes of types work well, and the lesson is that a different mix is better for a different time. The Creators will want and need to build new things: either extending the product in critical ways or on new projects and new products. If you don't have these projects, they may drift away. Find them a home on a sister team if you can! The Optimizers won't always be comfortable starting up a team, but you will need them later.

The natural result of this is that the team that made the product won't be the team that keeps it going year after year and eventually retires it. The camaraderie of the founding team won't keep everyone together forever.

These are generalizations, of course, but I would be quite surprised if you don't recognize your engineers and peers as falling into one or the other of these two categories. I am not sure why any particular engineer becomes a Creator rather than an Optimizer. I suspect it has something to do with a personal desire for or ability to deal with the unknown. During a project lifetime, there's a lot of "unknown" up front and a lot less later.

Balance

All right, you're going to hire high-capability engineers—but who are you looking for specifically? An explicit plan gives you a framework for evaluating candidates against your real needs. You can always hire any given high-capability engineer, but hiring to support a balanced team aims the team toward overall high function.

The criteria I use to understand my team and determine what sort of engineer I most need are leadership, experience, style, and roles. First I chart what I have, then consider the consequence of hiring variants in each category. The ones that make the most sense, best filling gaps and complementing the team, are the ones I list in the empty pages of my talent portfolio and hire for.

Leadership

You should expect leadership traits in every engineer you hire, but some stand out. There are engineers who can be the driving force in getting a team to grow through exemplary work, mentorship, and inspiration. (These are fantastically valuable.)

As a manager, you have to express every aspect of leadership, and you must also drive the team with those aspects that do not emanate from any engineer on the team. Too many of those gaps and you will drive yourself to exhaustion, so it's critical for your health and the team's that you keep your team staffed with leaders.

Leaders also serve a purpose in the recruiting process, as engineers have a pronounced preference to work with strong, leader engineers and will not only respond more favorably to offers but rise to the occasion during interviews and are more likely to get offers in the first place.

Experience

Like many people, I heavily discounted the value of experience until I had some. At this point, I hire the best engineers available, however much experience they have, and I don't worry about hiring developers with more experience than I estimate I strictly need.

The question I ask myself is, "Can I hire less experienced engineers?" The answer depends on the existing makeup of the team. If the great majority of the team—or all of it—is composed of engineers new to their careers, then I am better off focusing my energy on identifying highly capable engineers with more experience, rather than less. The team members are better off as well, because engineers have a tremendous ability to learn from each other, and, in my experience, greater differentials in knowledge and skill generated by experience often drive faster and deeper learning.

Experienced engineers as a rule are more capable, but it's not a direct relationship and it's not universal. While hiring, I focus on demonstrated performance, so it's not very important to me how much time an engineer has put into building their capability to their current level. I care about the performance level and worry about experience only to the extent that I try not to staff my team without any.

Style

Your team needs a variety of perspectives and styles to remain healthy, perform at peak levels, and improve over time. However, the nature of your work may call for differing attitudes and perspectives—what I think of as work styles. Knowing and taking advantage of your team's work styles will help you hire effectively in the first place and keep people interested and engaged over the long run as well.

Though you need a mix of styles, it's sometimes a challenge to make everyone realize that all styles are valuable. Many people mistake stylistic similarity to themselves for competence. That is, if your style is like mine, then you know what you are doing; if not, you're a bozo who shouldn't be trusted. A manager's challenge and duty is to keep all styles alive and well at the same time.

Here are a few styles I have used to characterize my engineers and teams. You will think of more that are better suited to your experience and circumstances.

Cautious Style

Some engineers are naturally patient and careful, or have become so through habit and training. They excel at thinking out and planning projects in meticulous detail. Every aspect will be considered and debated: reliability, security, and long-term maintenance. These engineers are particularly useful on projects involving financial transactions or core, critical systems that are essential to your business, such as databases that are used by many systems. They are risk-averse.

Quick Style

Engineers can be fast and furious builders. They may leap into action, building first and asking questions later. Frequently they are not too concerned about understanding every project detail up front, knowing they can check and correct the course quickly. They are not risk-averse.

Adventurous Style

Exploration and the excitement of newness can be fundamental drivers for some people. The software engineering profession has continued to evolve quickly over the past few decades, and that rapid change combined with its relative novelty (as opposed to insurance or accounting, for example) can be a major draw for such people. They are characterized by bringing in new ideas and trying things out. Usually these are tools and languages—sometimes inventions. They are willing to try a new technique right in the middle of a project: to learn about it and see if it helps. They are not risk-averse.

Perfectionist Style

Perhaps too rare are engineers who insist on a professional standard set at an extremely high level. Usually this is in product and code quality. They insist on testing thoroughly; they write their own unit test frameworks for fun; and they explain to anyone who will listen how they managed to get the project's code coverage up to 99.5 percent. The last 0.5 percent keeps them up at night.

It's All Good

I want to stress that these styles and more are *all good*. Sizing up your engineers and deciding what style you need to search for next to extend and complement your team will give you another tool for making great decisions and great hires.

Broad and Narrow Roles

What constitutes a balanced team depends on your organization, needs, and preferences. As an example, you may need test engineers on your team, or you may have a close collaboration with a team of specialist test engineers so you don't have to hire them onto your team.

Bigger teams usually start specializing certain roles, such as build engineer, tool engineer, or user interface (UI) developer. Arguably, civilization as we know it was brought about by the ability of a steady food source to support the creation of specialist roles at the dawn of agriculture—giving us room to become bricklayers, mathematicians, and so on. So there's a lot to be said for the power of specialization to create useful deep expertise.

However, narrowing the scope of a role too deeply will cause narrowed vision for your team and rob you of flexibility and some growth opportunities. A narrowly defined role limits the market you can recruit from and implicitly preempts your ability to hire great generalists. The likelihood is that your organization can be *more* successful with *less* specialization.

Team Flow

As Heraclitus said, all is change. Even a team that looks the same is changing from day to day. It learns new techniques and new modes of operations, experiments with small and large process changes, and responds to new information and new demands.

A team changes in a more obvious way as people come and go. There is an unavoidable rate of attrition caused by entropy, and you will eventually lose team members, however satisfied they are. The attrition rate is predictable

across a large group of people, but under macro circumstances—in the scope of one team—there is probably no truly reliable method of predicting how often people will leave except by extrapolating from recent history.

Still, you know there is a rate. You can guess at it from available information, drawing on team history or the organization's rate. Then you can use that information to predict, broadly, how frequently you will need to hire to keep the team at a steady size. My recommendation is to hire ahead of the rate and stay one engineer "above headcount," but that's not always practical.

It may not always be necessary to replace engineers. If the team is growing in strength and overall capability, but the work level remains steady, you may well be able to handle your workload with fewer staff. That's not the general trend these days for at least two reasons. First, it seems that there is always more to do and you can take advantage of any level of capability available. Second, the accumulated work product of a team can drain its productivity to the point that you need more staff to maintain and extend a product over time.

That second effect is the natural product of the support load that products generate and the accumulation of technical debt. A great overview of how that happens and methods for sidestepping the problem can be found in the paper *Nobody Ever Gets Credit for Fixing Problems that Never Happened* by Nelson P. Repenning and John D. Sterman.[1]

All change is stressful, and losing a team member is traumatic—even one for whom no one had special love. As a manager, you can help the team adjust and stay on course by keeping the big picture in mind: every team is dynamic and changing.

Building a New Team

Creating a team from scratch or from a small core of engineers is a wonderful challenge. The software engineers you should hire will be almost entirely Creator types, not Optimizers. These are the sort of people who make their own startups or build them from scratch with colleagues. Startups are a great place to recruit them as well—at any given time, many startups are leaving the inception phase or failing.

The first couple of hires will form the core culture of the team, so it's especially important to hire the right people. You need engineers who are leaders and great communicators, who inspire new team members with their technical

[1] Nelson P. Repenning and John D. Sterman, "Nobody Ever Gets Credit for Fixing Problems that Never Happened: Creating and Sustaining Process Improvement," *California Management Review* (2001), 43, no. 4. Available at http://web.mit.edu/nelsonr/www/Repenning%3DSterman_CMR_su01_.pdf.

acumen and welcome and mentor them. Poor hiring choices would include a dour or pessimistic engineer, who can kill your team and your project from the beginning, or a generally incapable engineer who can set the technical direction of the team going the wrong way, propagating large-scale errors that would be difficult to correct later.

Expanding an Existing Team

Most engineers are hired into standing teams, so that's the focus of this book. However, before you hire you should consider the cost carefully: the process of searching will consume your time and your team's time, and onboarding the new engineer will consume even more time. Short-term productivity loss is inevitable; the new team member will start with a net negative impact on team productivity. The team's social structure will shift and adapt to the new member, and the gelling process takes a setback (or starts anew).

Chapter 9 treats the impact of onboarding and is worth a review before you set out to hire more engineers.

Can You Hire?

Before you embark on a labor-intensive effort to find the right people, you must verify that you can hire them. You need to be able to pay the costs of conducting the search, compensation for the new employee, and various overhead expenses connected with any worker, such as insurance, equipment, and so on.

Chapter 3 discusses the cost of hiring and how you may go about measuring and optimizing it. Accounting for overhead is outside the scope of this book. Here I talk about compensation and your ability to make compelling offers to candidates.

If you find you can't afford to hire, focus your effort on making your existing team more effective.

Budget

Organizations vary greatly in how they allocate headcount and account for staffing costs, at least at the front line. Sometimes you get a number (e.g., three engineers); sometimes you get a set of job levels or classes (one senior engineer, two journeymen engineers); and other times you get a dollar figure ($340,000/year).

Whatever way you account for, fight for, or track the budget for employing engineers, the fundamental recruiting ideas do not change. Start by hiring the very best engineer you can afford, and keep hiring until you run out of money.

Compensation

Your task is to pay developers the right amount, bounded at the bottom by local market conditions and at the top by the danger of creating golden handcuffs.

Expense

Though you and your team can offer them a great many immaterial rewards, such as companionship and training, engineers are also going to want money. On the whole, they are excellent mathematicians and quite astute regarding market conditions, so they frequently want very good compensation. You must be able to afford engineers if you are going to pursue them seriously.

When you find the great engineer you want to hire, because she is great she may well ask for top-of-the-market salary, a sign-on bonus, and other cash rewards. It is perennially tempting to save money by hiring people at lower costs, but that approach in itself has substantial costs.

Paying less than top-of-the-market rates will have several undesirable effects. It will prevent you from hiring highly compensated engineers regardless of their fit, skills, and overall capability. It will tend to direct your hires toward the lower end of the market, which as a general but not universal rule has less capable engineers. It directs you toward less experienced engineers, who, no matter how capable they are, will not bring the hard-won lessons that experienced engineers bring with them. It can make you look and feel like a cheapskate, and it will discourage external recruiters from working with you because there are likely to be more lucrative clients hiring.

Losing a great candidate to a disagreement on compensation is a poor experience for both of you—after all, both you and the candidate went to considerable trouble to find each other, or at least you certainly did. That's unfortunate if the candidate wants an essentially unreasonable amount of money, but losing a candidate to a perfectly reasonable compensation request is heartbreaking.

Price Sends a Signal

You need and demand greatness. Offering high compensation informs candidates in unmistakable terms that you expect great things.[2] You tell candidates with dollars up front that you believe in their competency and the large impact they will have on your customers and your business.

---

[2]Valarie A. Zeithhaml, A. Parasuraman, and Leonard L. Berry, *Delivering Quality Service: Balancing Customer Perceptions and Expectations* (New York: Free Press, 1990).

The Use and Misuse of Golden Handcuffs

Golden handcuffs are levels of compensation high enough to substantially discourage employees from resigning. Companies frequently offer golden handcuffs to key members of teams and companies they buy out, so that the expertise they "bought" doesn't immediately disappear. Frequently these are time-triggered bonuses that mature in months or years.

Another kind of golden handcuff is not explicitly offered but occurs in regular compensation, intentionally and unintentionally. In the intentional case, the idea is that if you pay someone enough money they will not look for or be interested in any offers from competitors and startup opportunities. Unintentional handcuffs are formed when standard compensation packages become worth much more than originally intended. For instance, a stock plan that offers regular, timed stock disbursements may turn into handcuffs if the stock becomes worth much more than anticipated by compensation analysts.

In any of these cases, it becomes painful to leave. Retention of capable employees is a priority to companies because of the many obvious benefits, but sometimes attrition is necessary and good. When an employee is in a situation in which they would be interested in a new job *except for the handcuffs*, this creates a stressful tension. Situations that would otherwise be unbearable and lead to resignation, such as transfer to an inexperienced and irascible manager, don't resolve in the expected outcome. The employee stays on, and stress mounts. If you've seen someone in this position, you know that it is a bit torturous for them. They become unproductive and unhappy, and that unhappiness spreads.

In this case you now have an employee you should replace, because he or she is unproductive and damaging to morale. Firing the person is a lot more expensive and troublesome than if the natural progression of circumstances had led to a cleaner turnover.

I'm not saying you should never use golden handcuffs, and you may well run into the situation unintentionally. What I am saying is that when golden handcuffs are applied, perhaps through no fault of your own, you must be especially vigilant to monitor the employee's stress level and take early action to reduce it or help the employee move on. If you don't, you will be hurting—for paying your employees *too much.* Strange, but true.