HIRED FAST

THE JUNIOR DEVELOPER'S GUIDE TO GETTING A JOB

BY KEVIN KO

For my parents, teachers,

and mentors.

Table of Contents

1

CHAPTER 1

The Perfect Candidate

The job market can be unforgiving.

Succeeding requires focusing on the right

strategy: becoming the perfect candidate.

Introduction

Every so often, we hear about programmers who are so talented they'll never have to interview for another job. They get ambushed by top companies, all trying to offer them better pay and perks than the next company. "They probably don't even know what a résumé *is* anymore," you grumble as you revise your CV for the third time this week.

As junior developers, we aspire to be that skilled one day. Until then, it's back to writing cover letters and hoping this morning's recruiter tells us we've made it to the next round of interviews.

Looking for a job is a universally disliked endeavor and most people find solace in the fact that they don't have to put up with it too often. You go through the motions, sweat a bit, and hope it ends quickly and in your favor. Somehow the act of finding a job, a process that could have a huge impact on your future, seems a lot like going to the DMV to renew your driver's license.

In a world consumed by software and instant, open communication, the modern job interview sounds like an oxymoron. It's time consuming and emotionally toiling. Every rejection is time wasted and rent isn't getting any cheaper.

But what if you could tilt the scales to your advantage? What if you could drastically reduce the time it takes to get hired? What if you could get companies begging you to join even without being a famous programmer?

People are landing roles at their dream companies by simply changing the way they interview. It's not always about who you know, nor does it involve memorizing every algorithm in a thousand-page textbook.

If that's what you were looking for, you should turn back now; you're going to be upset by the lack of binary trees and big O notation in this guide.

Successfully getting job offers as a junior developer takes knowing the lay of the land. It takes knowing which buttons to press. It takes a little bit of hard work and creativity—words you're not afraid of as a newly-minted developer.

What I discuss in this guide is the framework for job interview success, from start to finish. It's a guide written by a developer for developers.

One thing I learned quickly into my development career was the value of stepping back and thinking about a feature before coding it. This way, I'd avoid reinventing the wheel when an open-sourced library already exists to solve the problem. Or I'd ask experienced developers how they'd build a feature, using their response as a springboard.

In a way, this makes me a lazy developer. I can't help it—I often work in high-speed startup environments where debating whether or not I should create my own CSS grid layout from scratch would be a major waste of everyone's time. I have well-defined ways of getting the answers I need so I'm always prepared when I work.

Unfortunately, I don't often apply this mode of thinking in other parts of my life. I *should* spend time thinking about my grocery needs and budget constraints. Instead, I go to the grocery store when I'm at my hungriest and I accidentally buy a few too many donuts.

I *should* have acknowledged that job hunting is a skill that can be improved and therefore mastered, but instead I dove into blindly submitting applications,

resulting in a lot of rejections I could have prevented. That was the first lesson I learned when I began my job hunt.

In this guide you'll learn from the mistakes I made as a junior candidate and the mistakes I see made by candidates today. Afterward, you'll learn how to truly stand out from nearly every other applicant. You'll be equipped with the knowledge that successful people, not just developers, have used to beat the odds when it comes to getting their foot in the door.

New Beginnings

In 2012, I was considering learning how to code when my friend linked me to a TechCrunch article about Dev Bootcamp, or "DBC" for short. DBC was the first of its kind: a short but intensive training program that taught people how to code—in other words, a vocational school on Adderall. This coding bootcamp advertised that graduates were receiving $80,000 salaries after completing their nine weeks of training.

I thought the best thing to do would be to supercharge my learning by attending DBC. At this point, DBC was still new, so it wasn't immediately obvious what I was getting myself into. Long story short, I decided to apply to Dev Bootcamp and was accepted into their February 2013 class.

I became a new person when I graduated and I could already tell that it was a life-changing, albeit exhausting, experience. While other people learning to code were spending years learning the same skills, I knew enough to be dangerous after only nine weeks.

I could build websites and solve problems with software. I was undoubtedly a programmer. Now I just needed to get hired to make it official, but it turned out that finding a job would be a lot tougher than I had originally expected.

At the time, companies were still unfamiliar with the bootcamp model, even in San Francisco. Despite the fact that the market for junior developers would only get worse over time, I still felt the pangs of being an inexperienced programmer looking for a high-paying Silicon Valley job.

Most junior developer positions were originally intended for computer science graduates. I feared that there was no way a company would hire someone with nine weeks' experience; what could I do to convince them otherwise?

After coming up short interview after interview, my fear slowly became reality. The statistics have since proven that the success story of graduating bootcamp and *immediately* landing an $80,000 salary, while not entirely untrue, was the exception rather than the norm.

Hired and Fired

Before bootcamp, I couldn't tell you the difference between Ruby and Rails. A few students in my cohort had had years of prior development experience and naturally coasted through Dev Bootcamp. They got jobs relatively quickly—that wasn't me. I was just a 24-year-old who loved the Internet, even if that only meant browsing reddit all day.

It took me three months after graduating to get my first offer. It was for an internship paying $50,000 at a small startup in Palo Alto. I was desperate after applying to forty companies with no end in sight, so I took the job.

Being employed was great, but it didn't feel like I fit well in the company. I needed much mentorship and hand-holding, but their limited staff didn't have the resources for that, so I struggled daily.

After two months of working, around the time of my birthday, I took a small weekend trip back home. It was my first visit home since I had moved to San Francisco. I felt uncomfortable with my current job and it was apparent to my friends and family given how excited I was initially starting my career. I returned to Palo Alto the next Monday and, to my surprise, was let go that morning (best birthday present ever!).

It felt like the past several months had been wasted. After three months of searching, I was only able to hold onto a job for two months. In other words, I spent longer looking for a job than I was able to hold one down.

I was jobless again, but I didn't want to make the same mistake of settling for a job that didn't fit me. I carefully planned out how I would find my next gig. I knew it wasn't possible to suddenly become a superstar developer overnight—I was a junior-level developer, there was no denying that.

But what if I could be the *best* junior developer? Was there a way to hack the interview process? What if I could hit all the right switches in an interview and get hired even without understanding esoteric programming concepts? There are companies that would hire me even if I didn't know how to implement quicksort, right?

Becoming the Perfect Candidate

I knew I could do better the next time I looked for a job. I consulted friends, mentors, salespeople, marketers, CEOs. The more I talked to people about how

they landed prestigious positions or won big contracts, the more I began to piece together a pattern.

I'd never really thought of applying to jobs as a skill before. I figured I just needed to cast a large net, get interviews, hope the interview questions were easy, and with a little luck, I'd get an offer.

It turns out that there are a lot of different ways to drastically increase your chances of getting hired. You have the ability to make a profound impact at any point during an interview process, from the moment you send your application until you receive an offer. I found that, once I understood the interviewer's thought process, the array of tools I could use to my advantage opened up and I started performing a lot better in interviews.

While I was hoping to find a shortcut, I found that becoming the perfect candidate was more of a mentality shift rather than following a set of tricks. For example, I read articles that recommended things like showing up 5-10 minutes early to make a great impression. I didn't like that advice: for one, rarely anyone is late for an interview so it wouldn't give me an advantage whatsoever, but more importantly, I was certain that wasn't the difference between a rejection and an offer.

Changing my perspective of the job interview process allowed me to employ more specific, high-impact techniques that got me more interviews and, ultimately, offers.

Getting Hired the Right Way

After getting fired, I knew the type of company I wanted to be at: Small, but with enough developers to interact with. Stable, but not too big, as I wanted to play an impactful role on the team. And of course, I wanted to be paid well.

I prepared diligently for a couple weeks and sent out five applications. Three interviews later, I had two offers, took the better one, and became an early engineering hire at a Y Combinator-backed startup called AnyPerk. Landing a role there was everything I had originally wanted and more.

You might have heard about the Pareto principle before, where 80 percent of the effects come from 20 percent of the causes. In your job search, I believe 80 percent of your immediate success comes from being a better job candidate, not from being a better programmer.

Here's why: As a junior developer, you've absorbed an immense amount of knowledge to get to where you are. Language syntax, design patterns, using version control, how to deploy an app, mastering your text editor, and so on. Yet the common approach for most job seekers at this point would be to read more technical literature and become an even wiser developer. It won't hurt, but chances are spending the next hour reading articles on Hacker News won't really improve your chances of getting hired.

So at this point, what offers us more success in a shorter amount of time? Becoming a better job candidate, not a better developer.

During my next job hunt, I didn't focus on improving myself as a developer. I didn't work on interview prep or learn any new languages. I certainly never dug into a textbook on algorithms.

Instead, I applied my perfect candidate mentality and I tailored techniques that I knew would greatly increase my chances of getting an offer. I submitted smarter job applications, hit all the right switches during the interview, and found different ways to make a lasting impression on my target companies.

Why This is More Important Than Ever

As a direct result of Dev Bootcamp's success, there are now over 120 bootcamps worldwide. DBC alone graduates about a thousand students per year.

The large number of bootcamps graduating an even greater number of students has saturated the junior developer market. This problem is worsened by the fact that enrollment rates for computer science majors in American universities has been on a sharp rise. Yes, there is a shortage of developers—*experienced* developers. There isn't a shortage of graduates (bootcamp or university) looking to earn Silicon Valley wages.

When I graduated DBC, the average amount of time expected for grads to land a job was in the 2–3 month range. Years later, it's looking more like 4–6 months, and with the barrier to entry for becoming a developer at an all-time low, that number will only go up.

The industry at large needs more developers but there's still a significant noise problem with many under-qualified developers vying for consideration. It's easy to accidentally blend in with the crowd if you're not careful. Popular startups can have a backlog of *thousands* of candidates at any given time, but your dream company might be looking to fill only *one* position. This guide provides the framework on how to land *that* job by helping you distinguish yourself from the rest.

Time is Money

There's no doubt that junior developers *eventually* get jobs—but when and where? The average pay for a junior developer in a large American city is $80,000. If this guide were able to help get you an offer even two weeks before you would have otherwise, that's an additional $3,000 or so in your pocket at the end of a year.

I believe that the most practical interview advice comes from those who have been in the same situation, ridden the same roller coaster, and have been on the other side of the interview table as well.

While this guide won’t source opportunities for you or make you a better programmer, you will learn how to be a better job candidate. The perfect candidate. This guide is what I wish I had had when I first graduated DBC. It’s the advice I'd give anyone regardless of their development background or programming language of choice.

As we’ll explore in Chapter 2, the path to becoming a better candidate first involves understanding the motivations of the company. Knowing how each interviewer thinks will give you a natural advantage over other candidates throughout the interview process as well as give you a framework on how to employ the techniques we’ll discuss in later chapters.

2

CHAPTER 2

Interview Intelligence

Don’t go into your interviews unprepared. Set up your application and interview strategy by understanding how your future coworkers think.

Know What To Expect

Succeeding in an interview isn't just about smiling a lot and saying the right things. Hitting it out of the park not only increases your chances but also puts you in a position to negotiate a higher base salary. You want your future employer to be counting down the days before you start! For them to do so, you must first understand how they'll be evaluating you.

A quick disclaimer: I'll be making a few generalized statements here which may not apply to all companies. Needless to say, a small startup's hiring process differs greatly from that of a large corporation. I'll mention the difference where necessary.

The Roles

Over the course of an entire interview process, you will likely speak with many people across different departments. Depending on the size of the company, you will interact with:

- A Founder
- The Chief Technical Officer (or leader of the engineering team)
- An Engineering Manager
- A Product Manager
- An Engineer
- An in-house Recruiter/HR representative
- An employee from another department

The Universal Hiring Guideline

Each interviewer is a chosen ambassador of the company, and the role of each is to represent the company in what is essentially a business transaction:

- The company has a need for more programmers, along with a general expectation of how much to pay for them.
- The company needs candidates who will be productive employees and not liabilities to their team.
- When hiring a junior, a company takes on considerable risk in resources spent training an employee who might leave or not pan out in the short-term (3–6 months), so there's a lot of emphasis on hiring the *right* junior.
- If a bad hire is made, it will negatively affect the company's culture, productivity, and bottom line.

This is the basic, robotic blueprint for interviewing and every interview stems from these principles. It's fairly straightforward, but we'll see how to tie it together throughout this chapter.

The Interview Isn't Just About You

It sounds counter-intuitive, but in your first interview with a company, they won't be looking to see why you would be a great addition to their team. Most introductory interviews, or "screens", will assess your likelihood of being a bad fit (culturally or technically), so they can quickly say "no" and move on. It's a reactive approach, which is necessary because a company can't devote engineering hours to interviewing just anyone who asks for a job.

Because a bad hire is so toxic to the company, *many* of your interviews will revolve around this reactive risk assessment, even if you're at the late stages of your interview process. The closer the interviewer's role gets to the Founder or CEO, the more this risk assessment plays into their interview style, as they have more to lose from accidentally hiring a poor performer.

In other words, the focus of the interview isn't necessarily on you, but rather your interviewer's deep-seated fears and anxieties. You might answer every question correctly but inexplicably get rejected anyway. That's the difference between doing well on an interview and crushing an interview. At the end of the hour, an interviewer can say to herself:

Bad:

"Okay, that candidate had good answers, but I still don't have a strong feeling about them. Plus, every other candidate today did just as well, I'll have to think about it for a bit."

Good:

"Wow, that candidate has the right attitude and just 'gets' us. They struggled a bit on this one question but they'll learn quickly. I can totally see them fitting well on our team."

What they say depends on how you approach the interview. You can either choose to answer their questions at face value or dive deeper based on what you think they're hoping to hear.

How the Founder Thinks

If your first contact with the company is with a founder, chances are the company is a small startup. They expect someone who can self-start, push back on implementation and product direction, and buy into the team culture immediately.

On the other hand, if you end up meeting the CEO after already completing several interviews with the company, it's likely a culture-fit interview for a startup with more than 20 employees (although you probably won't talk to the CEO of a large corporation). This is unlikely to be a dreadful, intensive interview once you've reached this point. You could have the company's core values memorized and have made mindful observations of the company culture (mission, vernacular, prevailing interests, dress code, and so forth), but I don't advocate blindly trying to push the right buttons. I'll discuss when you shouldn't pursue a company due to conflicting values in Chapter 6.

How the CTO Thinks

This will be similar to the founder interview in some ways, but it'll also be a much more technical conversation, unless the founder herself is deeply technical. The CTO will be trying to evaluate technical team culture fit: Does the candidate advocate the same development philosophies as the company? Will the candidate be a net positive for the code base? Does the candidate have a passion for software development? And so on.

For junior developer interviews, the CTO is looking for signs that you have the ability to learn quickly. Mostly so they can assess whether or not you can keep up with ever-evolving shifts in their technology and also have the potential to take on greater challenges over time.

Expect the most difficult interview to be with the company's highest-level engineer or technical manager, since that person is the one responsible for establishing the baseline for technical competence as well as team culture.

How the Engineering Manager Thinks

Engineering managers will be present in all but the smallest engineering teams, and their interview will be similar to the CTO interview, with a focus on your ability to integrate well with their specific team.

Managers collaborate with engineers to maximize productivity. In turn, engineers collaborate with managers to discuss problems. The engineering manager will evaluate your ability to communicate well and to resolve issues. For example, being able to push back on unreasonable expectations or requests when justified, or being able to ask for assistance quickly if you're blocked.

The engineering manager's ability to hire great candidates not only makes her own job easier, but in smaller companies, makes her look great in the eyes of company contribution as well. If you provide the sense that you'll eventually grow to become a hugely productive, well-fitting team member, the engineering manager looks great for having taken a chance on you.

How the Product Manager Thinks

Product managers typically separate themselves from the nitty gritty technical details and are more inclined to hear about your interests in the domain they're working with. For a startup, this could be the industry it operates in. For a large corporation, this could be the actual layer in the organization you'd be working with (e.g., internal tooling, dashboards, messaging).

In my experience, their questions are primarily behavioral about how you work within a team, what interests you about the company, how you deal with feature requests, development processes you're comfortable with, and so forth.

How the Engineer Thinks

A rank-and-file engineer is mostly removed from the financial ramifications of a bad hire but cares a lot about team fit, personal fit, and competence.

This interviewer is gauging your abilities against a standard interview that they've already given numerous times. Moreover, they're looking for someone they would be happy interacting with on a daily basis. It seems frivolous—getting the work done is all that matters right? Actually, how much you enjoy working with your coworkers has a direct impact on your productivity.

The engineer is used to interviewing unimpressive candidates who make lukewarm impressions, which inevitably lead to rejections. The key to interviews with engineers is to demonstrate a humble curiosity and an ability to learn quickly and effectively. Of course, your coding ability will be evaluated, but your interviewer might be the person you interact with the most when on the job, so interpersonal aptitude is just as important. I'll expand on this in Chapter 5.

How the HR Representative Thinks

This person likely doesn't consider technical team culture fit, which is usually a concept only the technical team truly understands. Many people outside of engineering don't have an informed understanding of what software development involves and are likely to make surface generalizations about what constitutes a good engineer. This lack of understanding isn't out of disregard;

engineering teams simply have their own processes and can't be quantifiably evaluated like a sales hire, for example.

HR/Recruiting staff aid in hiring across all departments. They will likely only ask you a list of standard questions to gauge your level of interest in the company, your overall attitude when interacting, your familiarity with their technology stack, and so on.

They might also be in charge of handling your application throughout the interview process. It helps to develop good rapport with this person—not only because they might be your future coworker but also so your candidacy isn't forgotten when they inevitably have to deal with hundreds of other applicants.

How the Non-engineer Thinks

These roles likely have even less consideration for engineering team fit than the HR representative, so they're evaluating you for culture fit. These interviews come up if you're invited to a lunch get-together or some other activity before you're hired.

While this may be disguised as a lunch or friendly get-together, this is definitely an interview, evaluating how well you get along with others in the company. Be sociable and open to interacting with anyone you encounter. Be polite, ask meaningful questions, both about the company, their work, or about what they like to do outside of work. Zappos, known for their friendly culture, even lets your shuttle driver get a say in your candidacy.

Every Junior Engineer Interview is a Risk Assessment

A bad hire, usually defined by their inability to coexist, *not* their programming ability, costs the company in a lot of ways: money, much time spent training another engineer, delayed deliverables, and damaged team morale. These costs are significant, especially to smaller organizations.

Hiring isn't just about finding the best candidate, but also making sure bad candidates don't slip through the cracks.

To sum it up, you will be evaluated on these metrics, typically in order:

1. **Level of competence**
    - Could you generate technical value quickly and without taking too much time away from other engineers?
    - Do you have the potential to improve beyond junior-level competence?
2. **Engineering team fit**
    - Will you consume your manager and colleagues' time and well-being with behavioral clashes or political drama?
    - Do you share the same engineering values as the rest of the team?
    - Will other engineers look forward to coding and interacting with you?
3. **Productivity**
    - Will you be an outspoken, creative, and intellectually productive team member?
4. **Company culture fit**
    - Will you be able to buy into the company mission and core values?

How to Read Minds

We saw the hiring guidelines earlier and reviewed how it changes for each type of interviewer. While *you* are the one being interviewed, the center of attention should be on your interviewers and what might make them reject you.

If you understand their innate concerns before the interview, you'll have an advantage over candidates who simply believe an interview is about themselves, and not the interviewer.

In reality, few employees get trained in interviewing or are taught how to ask the right questions. Even fewer are taught what to look for in your answers. Interviewing well isn't thought of as a skill that one could improve (like how applying to jobs isn't a widely-recognized skill), and many people remain novice interviewers throughout their entire careers.

So what if interviewers don't interview well? Hiring becomes a highly subjective process that gets warped as each company puts their own spin on it. In the worst case, companies violate employment and discrimination laws (intentionally or not). Interviewers are human and simply want to hear things that make them happy. They'll retreat back into their shells at the first sign of feeling uncomfortable.

So while you can do your best to answer their questions, successful interviewees take it one step further.

An engineer might ask you a list of programming questions, but she might really be worrying if you have what it takes to pick up their language of choice despite only knowing another language.

A product manager might ask you questions about working within a team, but she might secretly worry you won't enjoy working with their legacy code on their archaic stack.

A founder might talk about culture but her real worry is if you have enough passion for the company to endure the ups and downs of working at a startup.

Sometimes they don't realize these worries immediately, they'll forget to ask you, or they don't know *how* to ask you.

These worries are reasonable enough to predict, but if you're able to alleviate these concerns without them asking directly, you have the power of mind-reading at your hands. You'll begin to say all the right things and you'll suddenly become a strong candidate rather than a potential risk. The interviewer's attitude has just shifted from being reactive (gauging whether you'd be a fit) to proactive: envisioning *how well* you'd fit with their team.

As you begin to research the company and understand the nuances of their product and team, you'll have a good idea of what type of candidate they're really trying to hire. We'll go over research techniques soon.

To give you an idea, I've been on "the inside" of major tech startups that have needed to hire junior engineers, but specific traits they looked for included:

- Ability to work with sales teams and communicate feature requests with the rest of the product team. Engineers who happened to have a business background were favored here. (Booming software company that recently IPOed)

- Proficiency in a newer framework that the company was slowly shifting into. (This happens all the time at many different companies, in this case it was the technology-arm of a wealthy retail corporation)

- Ability to work independently, self-manage workload, and give reasonable estimations for feature turnaround (Small, funded tech startup in home cleaning).

If you're alert to the signs, you'll find out what the company is trying to do and what type of programmer they prefer rather quickly.

But Why Hire Juniors?

It's a time-sink to constantly interview, and there are a lot of ways for a bad hire to backfire, consuming even more time. Shouldn't companies always hire more experienced engineers instead?

1. Finding mid-to-senior-level engineers is very difficult

Remember, the company has a need—more engineering productivity—and hiring engineers, of any level, is a solution to that need. Whether it's a clumsy startup trying to grab any developer they can find, or a large billion-dollar corporation that simply needs a thousand more hands on deck, that need exists.

A large corporation wouldn't even need a thousand experienced engineers—there's a lot of procedural work to be done that lower-paid, more inexperienced engineers can fulfill.

2. Time-to-hire is shorter and initial costs are cheaper

A large corporation can fill a thousand seats with juniors much quicker than it would take to fill with mid-or-senior-level developers. Plus it's cheaper because it's a buyer's market for all but the best juniors (top-percentile Stanford computer science grads, for example).

In most cases, juniors are noticeably desperate so companies can offer any salary they want, knowing someone will take it eventually. Juniors are far less likely to have competing offers, so there's no bidding war there either. Interview candidate, send offer, receive acceptance, fill seat, and repeat.

Small startups looking for juniors are doing so in order to complement their senior-level staff. A startup deals with a million issues at any given time—the senior staff focused on those tasks don't have time for other important, but not as time-sensitive, issues that a junior could easily fulfill.

3. Homegrown talent is paid less overall

Both a junior- and a senior-level developer can hypothetically derive a million dollars' worth of value for a company within a month by implementing a new feature. The senior developer was paid more for her work, however, because her senior-level base salary is higher than that of a junior's.

Of course the senior developer offers more value to the company by writing better code, shipping quicker, handling more difficult tasks, and so on, but a company rarely evaluates their employees on these unquantifiable metrics. A manager doesn't look at lines of code written or deleted, commits per day, or number of monads used in order to determine that person's salary. That would be silly and developers would either riot or exploit it.

Instead, salaries usually correlate to perceived skill level and degree of responsibility, but there are times when salary can be influenced by frivolous reasons as well. Interviewing for a role specifically titled “Junior Developer” is one of those reasons—and your salary might be limited based on that title alone, regardless of your actual delivered value.

Imagine the junior stays for a year, improves substantially, and exceeds initial expectations. The junior's pay does not necessarily have to reflect the market value of an equally skilled developer right away.

Let's say you were hired at $80,000, and, a year later, your market value (how much you would reasonably be paid if you looked for another job) becomes $100,000. The company gives you a raise to bump your annual salary to $90,000. A 12.5 percent salary bump is generous indeed, and you will have happily accepted it, none the wiser. It may take another year or two before you're finally bumped up to $100,000. By then, your market value may have increased again.

Companies aren’t reliable when it comes to aligning salary with actual delivered value, especially in some companies where there might be a long, HR-driven process for increasing compensation. The sooner they anchor you to a specific salary point, the easier it is for a company to continue paying you that amount.

Research—The Why and How

How else can we shave off the “risk” label without already being a talented developer? You must **research** the company. There are a lot of application strategies you can employ, but each relies upon doing enough research first. Research isn’t just limited to clicking through their site and knowing what their

product does. Besides knowing what the company does, you'll want to know *why* they do it and *how*. I've found that the best resource for these is usually the company's engineering blog, but don't discount platforms like Twitter or even Instagram for an inside look on what the company is like.

The *why* will play a large part in whether or not you'll be happy at a company. It includes reasons like: "Wanting to improve how people get around the city," "wanting to help the homeless," "wanting to help businesses run certain processes more efficiently," or simply, "wanting to make a lot of money."

The *how* is even more important, and it includes the tools they use, whether or not they're an engineering-first organization (as opposed to sales-first, for example), what their company core values are like, and so on.

For example, Zappos is a large billion-dollar online retailer for shoes. One of their secrets to success is their belief that fantastic customer service allows them to build a familiar brand.

In turn they tend to hire those who are the most passionate, friendly, and kind, even for roles that aren't in direct customer support. This ensures the staff are all committed to the same values, and to Zappos, a cohesive work environment is also a productive work environment.

Finding this out about Zappos is easy—they mention their unique core values and company culture constantly in the "About" section of their website. For other companies, it might require a bit more digging, like reading through blog posts or seeing what kinds of outreach they're doing: conference talks their employees give, events they host at their office, projects they build for open-source, what their employees are tweeting about while at the office, and so on.

Preparing Our Strategy

Mind-reading comes easier when you know a lot about the company itself. Remember, when hiring, a company has a need to fulfill. What are they looking to accomplish with a new engineer from a technical standpoint? What are they looking for in terms of culture fit? Knowing the company well allows you to form powerful narratives tailored to the specific person you're interviewing with.

Research is the key to mind-reading successfully and consistently in your interviews.

If you were applying to Zappos, you'd want to weave your narrative to fit their core values. However, it's not enough to simply say you care about customer service or about creating strong team bonds; you must demonstrate it in novel ways, and we'll discuss how to do so in the upcoming chapters.

Now that we know what employers are thinking and how they're evaluating you, you can use this information to your advantage and prepare other strategies to become the perfect job candidate.

3

CHAPTER 3

How To Always Get a Call Back

First impressions are important. Here are some tips that will let you make the most out of your first contact.

“Not experienced enough.”

This is the single most common reason for a rejection, either before or after the interview process. When evaluating your candidacy, a company will sometimes decide to reject you because you lack *experience*. Experience doing what, specifically? And how do you acquire that experience without first getting a job? This reasoning seems vague and also kind of a catch-22.

Juniors aren't expected to be fully competent right off the bat—there's an understanding that hiring juniors means needing to train them. You shouldn't be evaluated based on your current knowledge but on your potential to be productive, yet employers still default to the “not enough experience” reason for rejection once they receive your application, *even if you would have done well in their interviews*.

When an employer receives a typical application from a fresh graduate, let's call him Junior Jeff, there isn't much to glean. Jeff hasn't worked on any significant technical projects before, so he’s unable to provide any insight on those. Jeff might have a personal website, but it's mostly a link to a hastily-prepared CV and a blog—though it's inactive and doesn't contain anything of technical worth.

So the employer checks out Jeff's GitHub profile and finds a bunch of familiar-sounding apps that she's seen on other student GitHub pages (i.e., “ToDo written in X,” “flashcards,” “blackjack”).

After some more digging, she finds a project that sounds promising. She checks it out briefly and can't easily discern what Jeff's involvement was. Maybe because the commit messages are unclear (“tests broke,” “update,” “I hate this API!!”) or because the application isn’t hosted online.

This is when the lack-of-experience rejection makes a lot of sense. Jeff simply hasn't had enough time in between learning how to code and applying to this company to have done anything worth discussing in detail, so, at best, Jeff looks like a hobbyist, not someone who can add significant value to a company.

It's possible that Jeff is actually a competent programmer, but scheduling and conducting interviews is time-consuming. All companies eventually have to draw the line between the people who will be offered an interview and those who won't. Sometimes this screening is done over the phone; sometimes it'll be based on your résumé and portfolio.

The Average Candidate

Junior Jeff and his type apply to companies worldwide and believe going the extra mile simply means changing a paragraph on his copy & pasted cover letter to look more personal. Jeff doesn't do any substantial research (even if it'd only take five or ten minutes) and inevitably sends generic applications to any company with an open position, believing that getting hired is simply a numbers game.

Here are two things that actually are numbers games: Nigerian prince email scams and recruiting via LinkedIn.

Programmers with a LinkedIn profile are probably familiar with the latter. Because you have "JavaScript" listed in your profile, you become a target for recruiters who comment on your "impressive background with Java" who think you'd be a great fit for a company looking for a "Senior Android Developer with 5+ years of programming experience." It quickly becomes obvious that the recruiter didn't read your profile; just 10 seconds of skimming through would

have clued them in on the fact that this position doesn't fit you at all. It leaves a poor taste in your mouth as you decide to ignore the email forever.

If that's how we react to generic recruiting emails, why do we send the same type of emails when applying to companies? Recruiters sometimes have to cut corners because their job *is* a numbers game. While they need to successfully place multiple candidates every year, you only need at minimum one job offer. So don't be the average candidate—don't send the equivalent of recruiter spam.

The Perfect Portfolio

Becoming a better candidate requires you to stand out from the average candidate. You now know what the average candidate does (send candidate spam), so standing out simply involves going one or two steps above that. The following is what the employer should have seen from Jeff's application. Not only would it guarantee Jeff an interview but it would also foster a positive impression that lingers throughout the interview process.

Portfolio Website

Jeff should have a portfolio showcasing meaningful projects. Each project should explain the problem it was solving or why it was built, how Jeff solved the problem, and with what tools. As a bonus, if Jeff has any design chops, the portfolio should exemplify the extent of those abilities.

Tip: When creating your own portfolio, it's okay to search for portfolio examples from individuals or agencies and heavily borrow design inspiration from them. Don't copy everything—get a sense for how people like to creatively showcase

what they've done and use that as a guide. You can see my portfolio here: https://www.kokev.in.

If you lack projects to show off, this may be a sign to start. If you don't have time to build side projects, consider still making a (visually pleasing) personal site to house your other creations, like blog articles or exemplary code samples.

Technical Blog

Jeff should have a recently updated technical blog about his experiences working on different projects, gotchas encountered, or other neat programming tricks he's learned along the way.

A blog isn't required. Some people dislike writing or the time commitment of keeping a blog updated, but this is an easy way to stand out for those who do enjoy blogging.

Tip: One very useful thing I wish I had the discipline to do is blog about every weird bug or error I've come across and how I solved it. Not only will this help you the next time you encounter this bug but you will also help others who search for the same error.

A blog is also useful because having a formed opinion on anything related to the ecosystem of development means you've graduated from simply being a copy & paste novice to someone who actively *thinks* about their code. This is a great signal for any prospective employer wondering if you're capable of contributing to the team in ways beyond just writing code.

It also pays dividends to sharpen your writing. Working in a team actually requires a lot of writing: when addressing an issue, discussing a bug, or debating a feature or implementation.

Sometimes you'll write to other engineers, product managers, designers, or even the CEO. Either way, demonstrating early on that you're a good writer implies that you're able to communicate well, and, in an organization, being an effective communicator is worth its weight in gold.

Résumé

Jeff should have a résumé that is easily readable and highlights important responsibilities he has had on technical projects in the past. In lieu of being able to talk about any technical heavy-lifting due to lack of experience, it should explain what Jeff's core competencies are in an honest fashion.

If Jeff has projects or past development work to include, they should be easily noticeable with clear, action-oriented descriptions. For example:

- "E-commerce website powered by PHP and Magento that served 5,000 unique visitors and 250 purchases per day with 99% uptime per year."

- "Website for ordering pizzas online. Personally designed and implemented all front-end interactions, including 5-step order flow and floating 3D pizza, resulting in 20% lift compared to legacy design."

But most importantly, Jeff's résumé is devoid of any spelling and grammatical mistakes, because those matters more than anything else.

Tip: Traditional résumés are slowly being displaced by other mediums, but many companies still rely on them to get an understanding of your experience

level and background. You'll still want a fantastic résumé though, it's another opportunity for you to stand out from the average candidate.

Be honest. This means not adding "Testing/TDD" to your résumé if you actually never write tests for your projects. Anything you put on your résumé is fair game for deep questioning by an interviewer.

Get your résumé proofread by a couple friends (native speakers if English isn't your first language), but don't spend too much time agonizing over the precise wording.

Make sure it's easy to read. A common mistake is using a cluttered stock template with bland typography because if it's hard to read, you run the risk of not having it read.

Chances are your development history is sparse, so put all programming-related line items as close to the top as possible and the rest (past careers, education, etc.) can go after.

Stick to a single page for the most part, it's unlikely that you'll have much to say as a junior developer anyway.

You can see my résumé here: https://www.kokev.in/cv. I used a template found on LaTeX Templates and I used a LaTeX editor like Overleaf to edit it, but there are many free résumé templates online that don't require LaTeX knowledge to use.

Need more help? Check out CareerCup's résumé guidelines.

GitHub

Jeff should have an active Github account page with his best, most relevant repositories immediately visible. Jeff should also provide specific code samples when given the opportunity.

Tip: The first thing anyone looks at when they look at your Github profile are your repositories. Github allows you to customize your pinned repositories, so make sure you have your proudest projects pinned.

When providing the person looking over your application a GitHub or Gist code sample of your best work, provide some contextual information, such as why you wrote that code the way you did. It saves that person a lot of time, is a considerate gesture, and gives you another opportunity to impress them.

GitHub Bonus

If Jeff contributed to other open-source projects (especially those used by more than a handful of people), the perception of his abilities skyrockets. This is easier said than done though, but Jeff's experience might be different.

For example, when working on a project over time, Jeff will inevitably interface with a library to the point where he becomes very familiar with it and its implementation. There may be opportunities for Jeff to create significant pull requests that optimize a function or add a small feature and so forth, especially if it's a relatively unpopular library.

Tip: You *might* get away with doing stuff like simply editing documentation—fixing a typo or correcting grammar—but I would advise against it if your intention is primarily selfish. With some digging, it's easy to find the source of the contribution, so don't rely on it working too often.

Similarly I would advise against posting frivolous issues, just for the sake of posting issues. Issues count as contributions in the context of GitHub, but, once again, this tactic will be quickly apparent to anyone checking, so you won't get bonus points for it. Legitimate issues are great though!

For more information on how to get started with contributing to open-source software, I recommend this episode of Healthy Hacker, a podcast by Chris Hunt.

Additionally, @yourfirstpr on Twitter, a resource not many junior developers know about, regularly tweets open-source projects looking for quick fixes. This can be a great way to start filling up your Github portfolio and demonstrate your ability to code on projects that aren't your own.

GitHub Megabonus

Companies with stronger open-source roots and a community-oriented developer culture will sometimes publish their own libraries for public consumption. Jeff contributing directly to these libraries is a significant bonus, especially if he's applying to the company that maintains them. It immediately relieves 90 percent of the doubt in the interviewer's mind of whether or not he can contribute to their company—he already is.

Tip: Research your company and see if they have public repos under their GitHub organization. For instance, here are Groupon's public repos: https://github.com/groupon. You don't have to make a significant contribution, but taking the time to contribute in some fashion shows a large amount of respect to the company, which will definitely be appreciated.

"I Can't Do These Things!"

If your experience developing projects and coding in general is truly sparse, consider holding off applying for now and taking the time to develop your portfolio or the other items above. When I was applying for my first job right after graduating from DBC, I created an imgur.com clone that allowed people to upload images and share them.

Creating that application was fun and I learned a great deal. However it wasn't significantly challenging to build and the back-end isn't complex. You can check out this application's source code at https://github.com/k50/skuvo.

When I started job hunting again for the second time, I was determined to show off my back-end skills, so I built a site that helps people track schedules and airtimes for their favorite TV shows. It also wasn't the most challenging thing in the world, but I mostly built it within a week, and I got to play around with new APIs and architectural decisions. When I interviewed at AnyPerk, I was questioned on precisely this project and its internals. You can check out the source for this site, HedonismBot, here: https://github.com/k50/hedonismbot.

I built HedonismBot because I identified a weak spot in my applications: I wanted to do back-end/full-stack development but my portfolio and imgur.com clone site suggested I was *only* proficient in front-end and a bit of design. Remember, understand the psychology of the employer, predict their initial fears and doubts, and alleviate them as early and thoroughly as possible. In my case, I built a back-end-oriented side project before applying again, which gave me something substantial to talk about.

What's Currently Lacking in Your Portfolio?

A common example is applying to a company that uses another language and framework than what you're used to. You can build a dummy app using that framework so you preempt the immediate worry of, "Well, you seem to know X but we're concerned you might not be comfortable with Y, which is all we use here." It's one thing to say, "I'm a fast learner and can learn new technologies on the fly." It's another to actually demonstrate it.

If delaying your job hunting process seems unfathomable, create two lists: dream companies and it'll-pay-the-bills companies. Apply to only the latter group while you develop your portfolio. Apply to your dream companies afterward. The benefit here is that you'll also hone your interviewing skills over time before applying to your dream companies.

By the way, some of you may have looked at the source code for my older projects and saw the chicken scratch of a complete novice. No surprise there. I was a bright-eyed, naive developer back then (still am!). You don't need to build something *perfect*—completing a functional app in the first place is a significant merit all on its own.

Developers generally have trouble completing their side projects, so people assume junior developers on the job hunt are a lot less likely to be able to "ship" a project as well. Demonstrate that you have the discipline to carry out a task from start to finish, which will immediately separate you from the average candidate.

Side Projects

Even with a full-time job, I still develop side projects constantly and find that each one teaches me a tremendous amount—either about myself and my coding

style, or it improves my understanding of a technology or tool. I personally believe building new things is the fastest and easiest way to learn or improve your programming ability, as opposed to reading technical books or blogs.

You can never predict what a potential employer might interview you about or want you to build on the job, but that's okay. Development follows certain patterns and understanding those patterns well will help you regardless. For instance, before I start a new project, I always plan it out in this way:

1. *What is the problem I want to solve?*
2. *How can I solve it using software?*
3. *Which tools can help me implement the solution?*
4. *How do I program this solution in a manner that is readable and maintainable one year from now?*

This process doesn’t require any lines of code being written, yet you will likely repeat this pattern every week at a full-time job. This process can take days of planning and constant interaction with other people.

Being able to discuss how you tackled a past project in this manner demonstrates that you will be able to handle the rigors of development on a real team in a real business. Even if your side project was a social media app for turtles and your prospective employer builds enterprise-facing invoicing software, developing a feature or program always involves a significant amount of planning, which is a skill that can be carried anywhere.

Before an interview, you should be prepared to discuss items 1–4 above in depth for your last big project, even if it was a team project during bootcamp or school.

Demonstrating that you mindfully plan and deliberate a project before executing it is what separates the professionals from the hobbyists.

Defining Employability

While becoming a better candidate or developing your portfolio will help you get hired, it's not a complete substitute for actual coding ability.

Each company has their own benchmark for what they consider junior-level employability—the minimum level of competence you must possess for your candidacy to be considered.

How do you know if you're employable or need to work on brushing up on your knowledge before applying? It depends on the company you're applying to, but my general rule of thumb is: can you individually build a complete project in your area of expertise without relying on tutorials or Stack Overflow to make significant decisions for you?

Looking up syntax, documentation for an API, how to do that one tiny thing, or how to use third-party libraries are fine of course, but building a project that depends on a tutorial to build a majority of the system doesn't count.

My reasoning is two-fold: For one, having enough experience building all the pieces of a working app should prepare you for most interview questions on a language or framework of choice. Two, some companies will ask you to build an app as part of their interview process anyway.

For example, as a Node.js developer, do you know all the steps you'd need to take in order to build a real-time chat application from scratch? As a web app developer, would you be able to build a site that lets users sign in and post to

multiple social media accounts at once? As an iOS developer, would you be able to build an app that lets people map their bike routes?

I made these examples up, but some of you may already be thinking about which web socket events or front-end framework you'll need to use to facilitate real-time chat, the best way to set up a user schema for managing multiple OAuth identities, or how to use a combination of CLLocation, MapKit, and local storage to draw GPS-tracked bike routes.

For many others, building something entirely from scratch is a formidable challenge. When learning to code, a lot of us started off by taking something already built, manipulating some of its bits, and experimenting from there. It's tempting to rely on a blog post, "skeleton" code, or tutorial that does a lot of the heavy lifting for you. This is fine if you're learning something for the first time, but try to avoid that temptation for your main area of focus if you want to improve faster.

If you aren't sure where to start or you constantly get stuck with small implementation details (common tasks like rendering validation errors, retrieving data, caching, etc.), that's fine—you know which areas to seek help in.

Specialization vs. Breadth

Many people, especially those trained at bootcamps, equip themselves with a full-stack skill set and try to become proficient with multiple languages or frameworks. My opinion is that, if you have any doubts about your current programming abilities, you should focus on one concept.

Knowing multiple languages will generally not help you during interviews if you're relatively new to development. I've rarely had interviews evaluating my proficiency in more than one programming language.

If I interviewed a candidate for a Ruby position who knew Ruby, JavaScript, Clojure, and Objective-C, it would be nonsensical of me to decide to conduct a portion of the interview in Clojure, for example. On the other hand, not being fluent in Ruby while simultaneously being lackluster in all languages will not make you an employable candidate.

I recommend starting off as a "T-shaped" developer if you insist on being full-stack. It's fine to have enough knowledge across a breadth of languages to solve most small problems, but designate a clear proficiency in one area of focus.

Companies aren't looking for a jack-of-all-trades, they're looking for the least-risky hire. As a prospective hire, you want to demonstrate you can be a guaranteed value-add to at least one part of a codebase. If your goal is to land a job as soon as you can, choose a single path, and once you're employed, you can make time to improve your other skills.

4

CHAPTER 4

The Golden Ticket

A little bit of passion and creativity can help you stand out or even get you an offer immediately.

An Imperfect Process

It may seem like “culture fit” doesn't matter much in a developer evaluation but many interviewers inevitably ask themselves two questions: Do I like him/her? Do I think he/she will do well on the job?

The reality is that hiring is an imperfect process and is highly subjective. The first question—whether an interviewer likes the candidate—is more subjectively evaluated than the second question. We can use this to our advantage.

For instance, we can reasonably predict that most employers might prefer a cleanly-dressed candidate as opposed to one that showed up in a dirty t-shirt and gym shorts. In that case, we’d dress appropriately if we wanted to maximize our chances of getting hired.

This is no different from reasonably predicting that companies prefer candidates who show genuine passion for their company, as opposed to the average applicant who only seems to care about vacation time and getting a paycheck.

Most small companies and tech startups have only a vague idea of what they consider to be the ideal candidate. It's like going to a buffet; you don't have your mind set on anything in particular, so you end up making your decision when you see the choices in front of you. It's a flawed system, but this is where you can shine.

Small organizations—whether the company is early stage or the engineering team itself is small—thrive on the creativity, grit, and proactivity of its employees. However, Junior Jeff applied to companies through a massive job listing aggregator, submitted bland cover letters (if any), and hoped his now

improved portfolio and résumé would do the trick. In some cases it might be enough, but not for Jeff's dream jobs.

The Instacart Story

Jeff's dream is to work at Instacart, a startup that provides on-demand grocery delivery services. He knows Instacart is a hot San Francisco startup and just finished raising millions of dollars from big-name investors. Brand name companies like Instacart generally don't have a hard time attracting junior talent.

In fact, popular companies like Instacart are so inundated with applications from unqualified candidates that HR immediately rejects people based on arbitrary metrics. Some candidates won't even get an initial interview! Don't have more than *n* years of experience? You might be automatically filtered out.

Jeff thinks he has no chance in this situation. Even if he gets an interview, he knows Instacart is wildly popular and that thousands of junior developers are applying there as well. How does he stand out amongst the bootcamp crowd, much less all the other junior candidates from top-notch schools?

Jeff is intent on shedding his reputation as an average applicant. He realizes that getting his dream job will require more effort than usual.

Being the huge Instacart fan that he is, Jeff has a great idea and orders an Instacart delivery for a case of beer. Only, instead of sending it to his apartment, he looks up the Instacart headquarters' address and sends it there, addressed to the CEO, Apoorva Mehta.

He also instructs the deliverer to attach a note that reads: “I'm Junior Jeff, looking for a Rails job with Instacart. Email me if you'd like to chat.” An hour later, Apoorva emails Junior Jeff about grabbing coffee and sets him up with an interview thereafter.

This is just one example of many where you can truly stand out from the rest of the competition—it's almost unfair. If this sounds incredulous, you can choose something else that fits your personality better (we’ll go over more examples soon). If you're still unconvinced about the feasibility of something like this, the story above is actually based on a real event: My friend Gordon, a designer, applied to Instacart this way and was contacted by Apoorva almost immediately.

Gordon was actually inspired by a similar stunt Apoorva himself pulled on Garry Tan, then one of the partners at Y Combinator, which incubated Instacart. Apoorva credits this event with the sole reason why he was able to get Instacart into Y Combinator in the first place. Gordon certainly did his research.

Golden Tickets

In the book Charlie and the Chocolate Factory, only a golden ticket will grant somebody access to the secret factory, so I call these displays of effort and creativity golden tickets—small, but significant examples of your clear passion for the company that will open doors for you.

Maybe a golden ticket like this won't be appreciated by a company with hundreds of engineers, but a company like that is also used to the bland cover letter and résumé combo that is delivered in droves—it’ll be easier to stand out anyway. In a bit we'll go over other tactics that are more BigCo-friendly that can still allow you to make a profound impression.

Why it Works

For one, 99.9 percent of other applicants aren't doing this. Ever.

Standing out from the rest of the crowd and being a memorable job candidate is going to drastically improve your chances of getting hired. Imagine if a company had to decide between two candidates for a position—you, equipped with a golden ticket, and someone else who might be more technically proficient, but didn't do anything out of the ordinary in their application. Your golden ticket makes for a very compelling argument in your favor. Candidates like you don't come around often.

Savvy companies don't hire engineers simply to write code; they hire engineers to make the company better. When first building a startup company, the CEO surrounds herself with talented and passionate team members. She sees how much productivity a handful of these types of eager team members can muster and wishes every future hire can be as productive. Strive to be the type of employee who practically could've started the company with the CEO. A golden ticket will help demonstrate that.

Best of all, demonstrating passion for their company in this unprecedented way might only take you a small amount of time, meaning there is virtually no cost to using the golden ticket approach.

When I applied to AnyPerk, my golden ticket wasn't anything clever like Gordon's. Instead, in the course of my research about the company, I discovered that their site had a recurring graphical glitch: their site footers weren't "sticky" against the bottom of the browser window. It's a common occurrence to anyone

who has built modern web interfaces before, but the solution isn't immediately obvious to those who haven’t.

So after my first engineering interview, I emailed my interviewer and future coworker to thank him for his time. I also pointed out the (non)sticky footer issue and linked him to a demo on Gist (Github’s code snippet-sharing service) with the CSS changes, specific to AnyPerk’s site, needed to fix it.

He was very thankful for that, and, while I don't think that was the *only* reason I got hired, it definitely helped shed a positive light on my candidacy. It demonstrated that I had experience fixing broken windows, that I care about user experience, and that I could diagnose a problem and come to a solution independently.

In reality it only took me five minutes to do. Five minutes to effectively seal the deal on a full-time job offer. I didn’t even break a sweat or have to pay for beer.

I later found out that I was considered over two experienced developers for the position. My interviewers believed I would make the best culture fit.

Very few people use golden tickets when they apply, but its return on investment is incredibly high. If you do hear about a remarkable job application, you might pass it off as something you would never be able to replicate or something too corny to attempt yourself. However, I’ve shown that your golden ticket can be something as simple as a one-line note in an email linking to a five minute fix.

Over a year and a half at AnyPerk, I interviewed developers for a back-end Rails position. Only a single candidate, out of the thousands that applied, ever went out of their way to impress us during the interview process.

We gave the candidate a lot of rope for that, but ultimately they weren't a fit for the position because their expertise was front-end-oriented. When we eventually had front-end positions available, we reached out to that candidate, but they had already taken another job by that point.

How often do you bomb an interview due to nerves? Or because an easy question threw you off and now you feel like you've misrepresented your abilities? Ever wish you could get another chance and prove that your poor interview was a fluke? A golden ticket that demonstrates your actual ability and creativity might be enough to give you the benefit of the doubt and land you another interview.

Famous Golden Ticket Examples

Loren Burton and Airbnb - Loren, a full-stack developer, created a page devoted to his passion for Airbnb and what his qualifications were. While Loren never ended up working at Airbnb, his page got significant attention on sites like Hacker News which led to interviews at other companies.

Nina Mufleh and Airbnb - Like Loren, Nina created a custom website dedicated to her enthusiasm for working at Airbnb. More importantly, she demonstrated why she would be a good fit, outlining a sound business plan about Airbnb's market potential in the Middle East. She was given an interview shortly thereafter and was even endorsed by the Queen consort of Jordan via Twitter.

Philippe Dubost and Amazon - Philippe, a product manager, was passionate about getting a job at Amazon, so he created this look-a-like Amazon product page to act as his CV. It went viral and he received hundreds of offers thereafter.

Tristan Walker and Foursquare - Tristan emailed Dennis Crowley, Foursquare CEO, eight times before finally being able to meet with him. Tristan was clearly passionate about Foursquare and his ability to provide value, so Dennis asked Tristan to sign thirty businesses to Foursquare as a test. Tristan found over three hundred leads in a week and was hired as the third employee.

Andrew Kim and Microsoft - Andrew, a designer, spent three days creating his own Microsoft brand redesign and posted it online. He received an enormous amount of attention, along with offers from many other tech companies. After he completed school, he signed as an employee of Microsoft and currently works in their Xbox division.

Feross Aboukhadijeh and YouTube - As a student, Feross developed his own instant-search version of YouTube in three hours. YouTube Instant became wildly popular overnight, and millions of people had used his project within ten days. He received a job offer from Chad Hurley, then-CEO of YouTube, but Feross turned it down to finish college. He took a position at Quora after graduating.

More Personal Examples

Eventbrite

I once applied to Eventbrite because I found out their office was right next door to my apartment and I was fascinated with having a one-minute commute.

I took the HTML and CSS of a typical event page on Eventbrite and edited it so that the content wasn't an event but rather included my developer biography and a ticket to interview me. While I wasn’t aware of it at the time, it was similar to what Philippe Dubost did with Amazon. Everything looked the same, except I

used my own photos and images where appropriate (it wasn't as narcissistic as it sounds, I hope). I gutted the payment portion of the page and instead added a few lines of JavaScript to render a link to my email address if the viewer "confirmed" themselves for this faux-event.

I pushed this page to my personal web host, found the work email of an Eventbrite senior recruiter, and emailed her the link with a personal message. She respectfully declined my candidacy, as they were only looking for senior Python developers at the time (I had no experience with Python), but she was very much amused by my efforts.

It wasn't a total waste—It was nice practice and I got to spend time with an established company's CSS code. It took about an hour in total.

Uber

While employed at AnyPerk, I developed a Chrome extension for the Uber website in my spare time. It's defunct now, but it allowed people to look at in-depth spending data on an old uber.com trip history page. I built it so I could get a sense of how much I was spending per month on the service. I decided to showcase the project on HackerNews, and, to my surprise, it received a significant spike of attention for a couple hours.

That day I was contacted by two separate employees about working with them at Uber. I wasn't looking for a job then, but you can easily see how this would have been a great way to establish a positive impression if I were.

When to Use This Tactic

Performing something like this can give you a tremendous advantage in the interview process, but it requires genuine interest in the company, plus a time commitment to research their product and what they need or could do better.

Since you might end up applying to a large number of companies, it's unreasonable to expect you to use the golden ticket method for every application. Out of concern for your time and sanity, I recommend golden tickets for:

a. Your dream companies.

b. When it'd just make a lot of sense. For example, building something using the API of a company whose core product is the API.

c. If the company seems on the fence about hiring you. For example, you've gone through several interviews with a company, but you don't get the sense that they're dying to hire you. Some icing on the cake can benefit you tremendously.

d. When the golden ticket can be done very quickly or easily. Sometimes you'll see something you can do and it'd only take five minutes. That's a great opportunity that costs you virtually nothing.

e. As a way to redeem yourself after a poor interview or relieve some concerns that were raised during the interview. For instance, if you had a decent interview with a company, but some sticking points came up about your proficiency in a framework that was unfamiliar to you, you could build something with that framework.

If you're the type of person who cares about improving yourself enough to read this guide, you're the type to be creative and passionate enough to figure out how to highlight your skills among the sea of other candidates.

Types of Golden Tickets

Here are some potential golden tickets to help get you started. Most of these work across the board, whether you're applying to a small or large company:

1. Fix something broken on their site

Bugs fall through the cracks all the time. Maybe they've overlooked how a certain widget appears in different screen resolutions. Maybe something looks fine on Chrome but not on Firefox. If you get hired for a web-related position, you will be interacting with the site constantly to patch tiny issues like these. Showing you can fix broken windows is a great sign.

Don't be dramatic about it; make it a friendly gesture at the end of your cover letter or email: *“Lastly, I noticed that some text on one of your pages becomes hidden in smaller screen sizes due to an element overlap. Your team is probably aware of the issue, but just in case, here's a Gist file of what I would modify in CSS in order to get it working properly.”*

Inspect their source code where you can and suggest ways to improve it. There are virtually unlimited ways to do this; there is always something broken or inefficient on any given site at any given time.

When logged into the Uber website a couple years ago, you could directly query your own rider account information from the browser console because the “User” object in that session was accessible via the global scope. This wasn't a

glaring issue, but users could see their own rider ratings (which were otherwise hidden from them), which got a bit awkward. This was a site maintained by some of the best JavaScript developers in the world.

Your golden ticket path will depend on your interests. If you're front-end and/or performance inclined, you can check out the web requests that are sent on page load via the browser developer console. From this, you realize they have certain bottlenecks preventing faster page loads, or they are loading large image files in their full resolution, which are shrunken down after the fact (a big performance no-no).

If you are security inclined, inform them of a possible vulnerability. It doesn't have to be something that requires a PhD in information security. For example, I've noticed on some sites that offer incentives for email referrals that the system doesn't check for email validity: you can simply enter fake email addresses and be rewarded. Companies are likely to know about the random bugs they have sitting around their own site, but bringing it up shows that you care about their products and success.

It might be difficult to find these, so I would only recommend it for those who don't mind spending a bit of time on the prospective employer's site. A longtime user of a service is going to have a better eye for what looks off. Are any of the sites you use often hiring developers like you?

2. Figure out what they eventually want to do, then build it.

Sometimes startups are very transparent about the challenges they face or the features they want to eventually roll out. They might blog (Yahoo actively endorses Node.js) or tweet about it, or you can make a safe guess given the industry and type of product. For example, Spotify is constantly trying to figure

out ways to get users to listen to more music. Uber is constantly trying to figure out how to get more drivers and passengers on their platform. Media sites are always trying to figure out how to attract more visitors and keep them on their site longer.

My friend Joseph Anderson, a mobile designer/developer, once inadvertently got a job interview with the CEO of ZeroCater, a lunch catering service for companies. Joseph figured that ZeroCater would eventually want to build a UI for ordering food directly on the site, so he designed a UI concept for that purpose. Joseph posted this as a blog entry with no intention of interviewing or getting a job. The ZeroCater CEO eventually found the post after searching on Google for his company's name and had reached out to Joseph personally for an interview.

Getting a response like this unsolicited shows how profound these golden ticket techniques can be. Get creative—what interesting things can Spotify do to get more people listening to music? What interesting things can your dream company do to get or retain more users/customers?

3. Suggest an improvement to an important part of their site.

I've been a member of Quora, a popular question-and-answer site, for a long time. Over the years I've noticed that their notification UI has changed many times, usually for functionality reasons. A simple blog post with your redesigned version of their notification system, even if unfeasible, shows some remarkable creativity and passion for their product. It'll help even if you aren't applying to Quora.

Publishing an opinion (even in the form of a design or implementation mockup) on something is a strong indicator of an experienced, savvy person. This is very much like our reasoning for maintaining a technical blog in Chapter 3.

You can do this with any site and as any type of developer. For instance, we could re-design or re-implement parts of Facebook in many different ways, depending on your background(s):

- Mocking up your own UI interpretation (design)
- Demoing an implementation of caching numerous, simultaneous "Likes" without suffocating a database (back-end)
- Propose your own algorithm to derive relevant posts to your feed (back-end, data)
- A better way to choose and select chat stickers (front-end and user experience)
- Accessibility improvements for the vision-impaired who rely on different devices for browsing (front-end)

The possibilities are endless, depending on your technical background. These are just arbitrary examples from one popular site out of many, what else can you come up with?

When attempting these methods, don't try to sound like you know better than the company; you might be holding on to some incorrect assumptions after all. Approach them with genuine passion, typically toward the end of the cover letter or in between interviews, such as, "I noticed you folks at Quora have been redesigning your notifications UI recently. You might be interested in a blog post I wrote a couple weeks ago with my own vision of it found here: [insert your link]."

4. Use their product in a way that gets their attention.

Companies want to hire people who are excited about their product, not just excited about making a salary. That you have experience firsthand with the product means you will have an easier time empathizing with users when it's time to fix a bug or implement a new feature.

We discussed leveraging the service with companies like Instacart and Eventbrite, but what about other types of companies? A community-oriented company, such as one with a thriving forum or message board, might be more inclined to hire you if you're an active member. I would imagine it'd be slightly awkward if I were applying to Twitter, Quora, or Medium without being a user myself. It might also be helpful if you used Quora to reach out to their hiring manager directly with a personalized note.

Elski Felson applied to Snapchat by creating a résumé story using Snapchat videos showcasing why he'd be a great fit and received a lot of attention overnight.

5. Create projects that directly help their users/customers.

Could you build something small that could help improve Airbnb's mission of providing unique travel experiences? Or help Zappos foster a friendlier company culture?

Developers at OfficeVibe created a game inspired by a Zappos quirk: whenever a Zappos employee logged onto their computer, they would be given a quiz about a random coworker. It was a small way for Zappos to build a more cohesive team environment, and OfficeVibe took it one step further and made it an online game that any Yammer-integrated company could play.

I thought this would've been a fantastic golden ticket example if used as part of a job application. They did their research, demonstrated creativity and passion for the problem, and they got to show off their technical chops. Since it used the Yammer API, it could have also been used as a golden ticket for a Yammer job application as well.

My Uber Chrome extension aligned with Uber's mission of providing a modern transportation experience. The archaic taxi industry doesn't offer software for analyzing your ride history and spending, but my tool helped you do just that through the Uber site.

If I were willing to build something to improve the Uber brand in my spare time, what beneficial things could I possibly create when actually paid and given company resources? Aligning my interests with that of the company demonstrates a significant cultural fit immediately.

6. Build neat projects with their API.

As a developer, I get excited every time a large company releases an API. There are unlimited things you could build once a large company's data is open to the world.

When Product Hunt, an aggregator of the best new products, released its API to the public, they coincided the launch with a massive global hackathon. Clearly Product Hunt is a company that values the "hacker"—they absolutely love it when people use their data to build new, useful products. If you want to work at Product Hunt, you are doing yourself a disservice by not doing something with their API.

Product Hunt isn't the only company in the world that loves it when users develop on their API. Stripe, an online payments service, has a very beloved API upon which many products have been built. You don't need to create anything original or viral; just build something, like a tool that emails you whenever a customer's credit card is about to expire.

Besides demonstrating that you like what their company is doing, API knowledge is valuable because you gain intimate understanding of that company's data model (and API, of course) before you've even started. It decreases the amount of time it would take for you to ramp up, and a lower ramp-up time also means you're less of a hiring risk.

7. Evangelize their products or services.

Even if you don't want to build anything, being an outspoken super-fan helps, as was the case for Tristan Walker and Foursquare. Be an actual member of their service and provide unique, valuable feedback. It'll also serve as a basis for being able to ask meaningful questions. Startups often hire their most vocal users early on and large corporations see it as a bonus that you value their service or product more than just their brand name (thus giving you a good answer to the age-old interview question "So, why here?").

A step-by-step blog article written to help others use a company's API to accomplish something will be greatly appreciated. Companies develop their APIs so that they can be used by other people, but the problem API designers face is that they might inadvertently build something that is difficult to use or is poorly documented. For example, a blog post that teaches people how to use Stripe to take monthly subscription payments or a post showing something fun people

can do with a company's service (i.e., using Twitter's API to build a train delay bulletin board) will be worth the hour or so it takes to write it.

Twilio, a platform for sending texts and other communications, constantly blogs about unique use cases for their API. One of their engineers, armed with the API, an Arduino, and some creativity, taught his dog how to take selfies. Like in our Product Hunt example, you're doing a disservice to your job application if you're applying to Twilio but aren't doing something with their API.

Which services do you enjoy using and/or consistently recommend to others? Which APIs offer you a lot of flexibility in building something fun and helpful?

8. Dedicate a custom site to them

Sometimes you might be stumped on what kind of useful widget you can build to show your enthusiasm for a company. I would personally have trouble thinking what I could possibly build for Microsoft, for example, since they're such a large company with multiple brands.

Instead build something that shows why you want to work with them, such as how well you would vibe with their culture and core values, how you can help solve the problems they're facing, and so on. No one expects you to single-handedly save their company, but knowing that you're a huge fan and have the resources to put together something like this is appreciated. It will probably be fun and allow you to be creative. Loren and Nina did this with Airbnb.

This isn't limited to just startups—most large corporations often have engineering blogs that discuss, to great depth, what they're working on, what tools they use, and what their challenges are. I mentioned Yahoo's enthusiasm for Node.js earlier. A quick search ("yahoo node") reveals a slideshow written by

a Yahoo engineer about what Yahoo builds on Node.js and what challenges they're facing.

Does anything on that slideshow resonate with you as a Node.js developer? You don't have to build a world-changing library that solves their problems indefinitely, but an amateur project that at least attempts to address those issues shows you've bought into the engineering mission at Yahoo, eliminating much perceived risk in hiring you.

9. Contribute directly to their projects.

As a designer, Andrew Kim created his own spin on Microsoft's brand—his interpretation of an improved Microsoft. As a developer, you can make improvements by directly contributing code, like we discussed in Chapter 3 on preparing your portfolio.

In Andrew's case, he was a really gifted designer, but that doesn't mean you must have a product go viral in order to get noticed. Some employees of thoughtbot, a highly-regarded Rails consultancy, were hired as a direct result of their contributions to thoughtbot's open-sourced projects.

While this may take a bit longer, how could anyone argue that you're not experienced enough to work there if you're already contributing directly to their source code?

10. Do something different.

Companies like to see signs that you didn't send the same application to a hundred other companies, so differentiating yourself with a golden ticket is the best way to indicate that. Golden tickets range from being incredibly simple, like referencing a talk one of their lead engineers recently gave at a conference in

your cover letter, to being time-consuming and complex, like building an entire project directly related to the company.

Regardless how much time you choose to spend, the end goal is to differentiate yourself from the average candidate. Some people won't have time to build side projects or invest heavily into golden tickets, which is totally understandable. The very least I ask is ten minutes researching your favorite potential employers through their blog or other media (and not just reading their "About" page) to discover their values and their challenges. If it's a good fit, finding ways to stand out from the average candidate will come naturally to you.

Sometimes a company might not be responsive to a golden ticket. Feel free to still use your golden ticket as a learning experience or a résumé booster. You created a tool for one payments company but they still rejected you? Their competitor won't hold it against you if you reference that project in your cover letter.

Cover Letters

When I first started applying, I wrote what I considered decent cover letters. I rarely resorted to copy & pasting except for a line or two. Out of forty applications, I received a positive response from about twenty (and over a dozen interviews thereafter). This was a healthy ratio (2:1, applications to responses) given I was a nobody with zero job experience at the time. I think you can do better though.

Showing that you're genuinely interested in working with the company makes the employer more likely to consider you. After all, you can teach someone how to reverse a linked list, but you can't teach them passion.

What do I write in my cover letters? In general I'll provide links to my latest project(s) or a list of what I'm mostly competent in. Then I'll include the most important part: reasons why I want to work for them and how I would be able to contribute, using what I wrote earlier as proof that I can actually deliver.

Have a good understanding of the company in order to set up your cover letter (remember, the why and the how behind what they do). If the opportunity arose, I would reference tidbits about the company that made it apparent I didn't just hear about them twenty minutes ago.

I once conducted an interview where the candidate referenced an old company blog post I wrote, which was a pleasant surprise. That seemingly benign reference had a deep impression on me *because no one other candidate had ever done that*. It made me believe the candidate was serious about working with us. You can use a similar technique for your interviews or your cover letters.

The bottom line is that the people reviewing your résumé want to feel like their company and their time is being respected. Once again it goes back to the cardinal rule of research.

Cover letters are a subtler, quicker way to show your passion for a company that doesn't require the extremes mentioned in the golden tickets section. For example, here was my intro/cover letter to AnyPerk's CEO when I first applied (my notes will be in brackets):

> *Hey Taro and the AnyPerk folks!*
>
> *My name is Kevin Ko and I'm a Rails developer with a year* [I rounded up] *of experience. I saw that AnyPerk had just started hiring* [this was true for

the engineering position, which showed I kept tabs on the company] *and wanted to reach out, specifically as a Ruby engineer.*

I started coding as a side hobby last year and I quickly realized I loved it and wanted to pursue it full-time. In February I attended Dev Bootcamp, a nine-week programming school, where I spent twelve hours a day sharpening my skills across the stack.

Most recently, I've completed a two-month internship with [redacted]. I was one of five developers serving over six million users, which provided an incredible experience working on a production-level app with other engineers in a rigid development cycle. [I would talk about a project here in lieu of any past work experience]

I'm excited to begin my next step in my already rapidly-evolving career as a developer and thought AnyPerk would be a great fit. On one hand, AnyPerk has grown incredibly fast but I would guess is still nascent enough for me to contribute to many parts of the code base. On the other, I'm a quick, ambitious learner and I'm not a stranger to startup life. I believe I can contribute to AnyPerk pretty quickly despite my relatively small amount of experience.

I'd love to talk more if you think there's potential for a fit. Otherwise, best of luck with AnyPerk!

Kind Regards,

Kevin Ko

I wasn’t aware of the challenges they were working on at the time, otherwise I would have referenced them directly and how I could contribute. I went the

route of convincing them to take the risk on me by emphasizing my experience in scrappy environments like DBC and my past startup internship.

I also made sure not to get too detailed, like providing links to my GitHub or talking about my projects on a technical level, because I knew Taro, the CEO, was not technical himself. Instead my goal was to convince him to get on the phone with me in the first place and then talk about projects when asked.

Tailor your cover letter to the person reading it. Sometimes you'll have no idea, and you'll have to assume HR is the one vetting your application. Other times, your first contact with the company might be the CTO who hands you a business card at a meetup and asks you to apply by emailing him or her directly.

In my example, I knew the email would go to the non-technical CEO specifically, so I made sure to keep the dev jargon to a minimum. As a refresher, you can take a look back at Chapter 2 on interviewer roles to learn what certain roles look for in a candidate.

5

CHAPTER 5

Mastering the Interview

Interviews may be gut-wrenching, but not for those who adopt the correct mentality, avoid certain red flags, and prepare for code challenges the right way.

Mentality

If you've followed the advice so far on succeeding before the interview, that momentum will carry weight into your actual interviews, but don't rely on it alone. At the end of the day, you need to crush your interviews and that usually involves being proficient at some aspect of programming.

There are many different types of interviews you'll be taking, and it'd be impossible to cover them all. However, keep in mind what we've already discussed as it pertains to each interviewer role. Besides your programming ability, your personality is also being evaluated. Try your best to assess what kinds of specific qualities they are testing you for. You can do this during the interview by noticing the types of questions they tend to ask or by getting a sense of the company's core values beforehand.

Dissolving Nervousness

Even senior developers get nervous during interviews. It's unnatural to be placed under so much scrutiny, and very few people ever get used to it.

The reason why you get nervous isn't because you're being judged though. Your roommate or pet probably judges you when you sing in the shower, but you don't care, right? The difference here is that you stand to gain a job from the interviewer, but your dog isn't responsible for anything except barking at squirrels.

When the perceived asymmetry widens between what two parties can bring to the table, the more one party will feel nervous and the other will feel more in control. In other words, think about a company on its last legs, needing more

money just to survive another month. For the struggling CEO, successfully raising money will be difficult because while their entire company is at stake, the investor doesn't have much to lose.

It's easy to think when you first start applying to jobs that you're just begging people to hire you out of charity. You desperately need a job, but they can always find another junior developer candidate.

Snap out of this mode of thinking. We discussed previously the reasons why companies *need* to hire junior developers in the first place—it's not just a charitable experiment. But that still doesn't help you, because we discussed earlier how the junior market is a buyer's market, right?

It's true, but there is still a shortage of junior developers who are humble, curious, and able to learn quickly. Your goal is to distinguish yourself as one of those junior developers.

When you're being evaluated in an interview, you're certainly being judged on how well you can code. But juniors are not expected to have memorized framework documentation by heart. They're expected to have got their feet wet, but, more importantly, to be able to learn at a fast pace so that it doesn't slow down the engineers assigned to mentor them.

Remember, every junior engineer interview is a risk assessment. That's why it's important to have a great attitude before, during, and after an interview. If I had to personally choose between two junior developers—one who had an open mind and was easy to get along with, and another with more experience but was hard to work with—I will always pick the former.

So don't go into interviews thinking as if you're begging a malevolent dictator for their mercy. The person you're interviewing with could be your future coworker, so make it a pleasant experience for everyone involved. As an interviewer, I believe one of the worst things about interviews is the potential for them to end up tense and awkward. Demonstrate that you're not a risky hire and go beyond that. Show them that you're someone they'd *love* to work with.

You won't be a fit for every company in the world, but that’s okay because not every company is a fit for you. Think about your favorite clothing line. Most likely you identify with a small subset of brands and not so much with many others. With a job there will be times when you find out that the company just isn't what you thought it would be, or you'd have to make certain sacrifices in order to work there. You are always determining whether or not a prospective company fits you, and it’s natural to not feel comfortable with a lot of companies.

Try pretending as if the company has already hired you and your interaction with your interviewer is your first day on the job. On your real first day, you’d normally have more enthusiasm than nervous energy because your mentality is no longer one of anxious begging but rather that of eagerness and curiosity. That’s a great state to be in during an interview.

Sometimes there's no helping the situation, some interviewers might turn out to be jerks, but a company that places jerks in positions of power is probably not a place you'd want to work at anyway. Try to have fun with your interviews and chalk it up as a learning opportunity regardless.

I’ve learned a lot about interviews while on the other side of the interview table. Try holding a mock interview with someone else, such as a friend who just

started learning how to code, where you play the role of the interviewer (you can find beginner programming questions online). The key here is to get a sense of what it feels like to sit with someone for thirty minutes uninterrupted and have them do 80 percent of the talking as you scrutinize their answers and evaluate their delivery.

It’s an unnatural social situation, but you’ll start to get a feel for what’s pleasant and what isn’t during an interview. In my experience, candidates who were willing to speak casually and openly with me during the interview seemed to do better than most.

The most successful interviews I've had (either an offer or close to that stage) were the ones where I went in feeling very relaxed, as if I were already friends with the interviewer. I made time to have an honest, casual conversation with my interviewers. Maybe we shared war stories about dealing with the same finicky bugs in the past or discussed what was interesting with the new version of Rails.

When you can approach your interviewer as a peer and not as a judgmental overlord, you’ll begin to feel more comfortable and your interview will go a lot more smoothly.

Tough Questions

Interview questions aren’t easy. It's hard to keep your composure when you get stumped, but I encourage you to maintain an open line of communication regardless. Most of the awkward silence comes from a code or problem-solving challenge that renders the candidate speechless for five minutes. During these moments, verbalize what's going in your head, even if it's entirely elementary.

The awkward pauses only stand to make you more nervous, which increases the awkwardness, making you even more nervous. If you can mitigate this, then at the very least you will prevent yourself from slipping into that cycle of nervous doom.

When I solve problems at work, I either close my eyes and think about it in silence like I'm Sherlock Holmes, or I pick up a pen and start diagramming parts of the puzzle. Since doing my best Benedict Cumberbatch impression has yet to impress anyone, much less an interviewer, in an interview I like to draw out the problem if I get stuck.

You may or may not have the same habits as I do, but you definitely have your "groove" in which you problem solve effectively. Adapt it to your interview, for example, by asking your interviewer, "If you don't mind, I'd like to diagram the problem on paper as I'm a better problem solver when thinking visually."

Breathe

Breathing has been shown to help dissolve anxiety, but we constantly forget to do this in the heat of the moment. Just remember to afford yourself the second to take a breath when you need to. A good breath is one that is calm and deliberate, but deep, that feels as if it extends to your stomach. There are other, more intense breathing techniques you can practice, but taking the simple deep breath described above will already help tremendously.

Coding Challenges, Algorithms, Puzzles

You'll often get coding challenges, such as "Sort this array without using the sorting functions built into the language," or, "Recursively implement this

solution.” These questions are asked to test not just coding competence but also communication and problem-solving abilities.

I'm not a fan of asking these types of questions personally. I prefer questions that more accurately represent what you'll have to do day-to-day, like fixing a bug, reviewing code, identifying performance bottlenecks, and so on. I’ve never been asked to develop my own sorting algorithm in my work.

Still, these questions are pretty popular, along with various abstract puzzles. The key to these is to practice them as often as you can and to identify the patterns they tend to share. They typically involve manipulating raw data or a data structure. You’ll probably have to loop over something. You can find a lot of resources to help you online, such as on Interview Cake or by reading *Cracking the Coding Interview* by Gayle Laakman McDowell.

The MacGuyver Method

The way I approach coding problems starts with understanding one simple pattern: Given X, return Y, which is also the basis of input/output, a fundamental of programming. Are you familiar with the 80’s show MacGuyver? In each episode, the titular hero typically builds a contraption using the resources within his vicinity in order to foil the bad guys’ plans. In other words, he takes X and makes Y. The steps to MacGuyver success are as follows:

Step 1: Identify X

Step 2: Manipulate X

Step 3: Create Y

Most generic coding questions can be solved with these 3 steps. That's because Fizzbuzz can be solved with these 3 steps and most coding questions are derivatives of Fizzbuzz. Here's an abstract of how you'd solve Fizzbuzz using the MacGuyver method:

Step 1: Identify what you're given (likely a range of integers) and what the output needs to be (printing a mixture of numbers and words).

Step 2: Manipulate what you're given so you can dynamically output words. You'll likely need to use some sort of iterator in order to solve most of these problems, including FizzBuzz.

Step 3: Ensure words appear when they should.

In interviews, I often see people immediately freeze up and get caught up in the particulars of the question.

First, understand what you are given and what you need to return. Next, condense the problem down to something more manageable. Your implementation might not be performant; don't worry about that unless you can easily devise something better. Just get something working as soon as you can.

Pseudocode or verbalize your thought process on how you would solve this without actually writing code, and use your notes to help you once you do start coding. This will also help prevent the initial wave of nervousness that strikes many people. Be honest when you're at your limits but be open to suggestions and to learning during the interview.

Putting it Together: A Sample Approach

Let's take an example question I've asked some friends when helping them interview.

> `Write a function that takes an array of integers as an argument and returns another array of the two highest integers in the argument. You may not use the language's built-in array sorting or array minimum/ maximum functions.`
>
> `E.g.,` **`myFunction([5,4,7,3]) // returns [7,5]`**

Here's a possible strategy to effectively tackle this question. Note that what's important here isn't necessarily writing the best possible solution to the problem but communicating your thought process and implementation:

1. As the realization sets in that you're being asked a dreaded coding-challenge question, control the urge to stress out by verbally repeating what has been asked of you and defining the function.

> *"Okay so it seems I need to create a function, give it an array argument, and it needs to return another array. I'll start with defining the function. Let's call it maxTwo."*

2. Verbally explain how you might solve this and do this every step of the way, even when you're brainstorming or actually writing code.

> *"This would be simple with the .sort() function, but I can't use that. It seems I'll need to find another way to compare integers with each other."*

3. If you're having trouble, keep pseudocoding and making sure your assumptions are correct. Ask questions if you need clarification. Condense the

problem into smaller pieces if necessary. Use the tool or practices that get you into your groove—in my case it would be diagraming the problem with a pen.

> *"I'll need to iterate through the array in order to compare integers with each other. I eventually need to find the two highest integers, but maybe it'll be worth it to restrict the problem to just finding the single largest integer first, and that'll help me solve the second part of finding the second-largest integer."*

4. We decided to make this question easier by halving the problem—finding only the largest integer as opposed to the two largest integers. Verbalize what you're about to write in code.

> *"So I'll create a variable, we'll call it 'largest_int', to store the value of the largest integer at any time. I'll create a for loop and compare the current element in the iteration with this variable's, value and determine which is larger. If the current element in the iteration is greater, I expect that it will replace largest_int's value."*

You decide that you will change a `largest_int`'s value whenever you encounter a larger integer in your initial array. Once you have that complete, remember step 1, your input and output. Our output here is an array, so we'll need a way to transform a primitive like 42 into an array that contains 42. In many languages, you can simply return `[largest_int]`.

Easy! Now we have to get the second largest integer as well. But we have a good idea of what the output looks like now, we simply need to adjust the code in the middle to return two specific values instead of one.

Keep verbalizing this thought process and your methodology until you've got a working solution. I know it's easier said than done, but try to be mindful about communication and asking questions when you get stuck or need to test an assumption.

Asking questions is very important, especially on questions that are open-ended, so don't just start coding right away. Use whatever materials they give you to jot down notes: for example, keeping track of the effects of the iteration at each index as you're progressing through it.

For the curious, here's a possible answer to the question.

Simplify

In the exercise above, I mentioned condensing the problem into smaller pieces. Instead of solving for the two largest integers, we halved the problem and approached it by only solving for the single highest integer first. Solving just for the largest integer is a much more reasonable problem to tackle.

Being able to simplify complex problems and solutions is a hallmark trait of a capable developer. Some companies have interview questions that target this trait specifically. Regardless, always take a minute to think about how you can possibly condense a challenging question or problem down to a more manageable scale. Diving right into code is usually the incorrect approach.

For instance, if the exercise above (which used `[5,4,7,3]` as an example argument) offered the argument `[58.97,24,17.4,93,63,8,59,0.51]` instead, you might feel intimidated while working through the question. It's hard to read, there are numerous elements in the array, and it includes decimals for no reason.

This is probably a red herring used to throw you off your mental balance. Instead, you can simply use the argument `[1,2,3]` as you solve the problem, going back to the long example array once you think you have a working solution.

It sounds like an obvious tactic but our natural inclination as newer developers is to immediately tackle a problem without stopping to think first. We expect coding challenges to be tough so when we face a question laced with complications, we think nothing of it and incorrectly accept the problem for what it is. Always try to simplify a problem first in a way that's manageable or familiar to you.

Computer Science for Bootcamp Grads

Many bootcamp graduates don't have much experience in fundamental computer science concepts because they're not taught that at coding bootcamps (in favor of having more hands-on, full-stack experience).

The interviewers asking you these are themselves programmers who graduated from computer science schools and were also asked the same questions when they first started their careers. To them, programming begins with these basic building blocks of data structures and algorithms, and they are correct. I find that the modern frameworks taught in bootcamps are high-level enough to abstract away most of the considerations you'd likely make if you were taught computer science at school with a lower-level language.

There are more jobs out there that expect you to know algorithms and computer science concepts than not. I leave the decision up to you on whether or not you think it's worth your time to acquire a strong grasp of these fundamentals. It's

definitely valuable, but there are always other things to learn that could make you just as employable.

I don't want to make a case for one being more important than the other, but each company has different interviews, and they look for different types of skill sets. Follow a path that makes the most sense to you, carefully considering the roles you're applying for and the types of companies you're applying to.

For example, It's likely that if you interview for a front-end role, you might not be asked about the space-and-time complexity of a function, but you still might need to show that you can identify when something may be a performance bottleneck in the browser. That something might just come from experience or reading blog articles, not necessarily from a computer science textbook.

If you interviewed with me, I would be more interested in learning the limits of your framework-specific knowledge, not whether or not you can sort an array efficiently. The type of activities that would help you in my interview specifically would be reading blog posts, listening to podcasts, and building projects.

Common Computer Science Concepts

There are a million questions you could be asked that involve computer science concepts. Chances are you will inevitably want to study these concepts if you didn't learn computer science formally—it'll help you throughout your career anyway, not just in passing interviews. For data structures or algorithms, I would begin by Google searching [*algorithm_name] in [your language choice]*, e.g.: *trees in Ruby*.

Here are some examples of common computer science concepts that come up in interviews:

- Big O Notation; understanding space and time complexity of a function
- Algorithms (searching and sorting are common ones)
- Data structures like arrays, hash maps, trees, graphs, linked lists, stacks and queues
- Anonymous functions, closures, callbacks
- Design patterns

If you're asked about something with which you're not familiar, don't give up immediately. Be open to working through the problem with the interviewer. Unless you're being interviewed by a robot armed with a checklist, most interviewers are not expecting you to get everything correct.

If your computer science fundamentals aren't great (or are non-existent) and you're being grilled about it in an interview, I wouldn't frame it as a hindrance. For example, I could say, "While I don't have a formal background in computer science, I still have an appreciation for fast, readable, functional code, and I'm open to feedback when my code isn't up to par." Lacking computer science knowledge isn't a deal-breaker. There are plenty of productive developers who don't have a strong grasp of these concepts, myself included.

It will still help to know why looping over a collection within another collection loop can be bad though. Or why you want to avoid instantiating a lot of data in memory. Don't be oblivious to computer science subject matter; be open to learning them, but be realistic about your shortcomings and be okay with not knowing everything (believe me, this feeling never goes away no matter how much you learn).

Common Practical Concepts

Conversely, if most of your programming education was traditional (i.e. a degree in computer science) and you have minimal experience working on production-level applications, the most difficult questions you might face are when they pertain to practical scenarios.

For example, how would you describe the database table relationships of a site like Instagram? How would you implement an upvote/downvote system on a site like reddit at scale? What happens under the hood when you go to google.com? Teach me something very few people know about JavaScript. Describe the flux (React) pattern to me.

Be willing to walk through the question to get to the answer. Don't worry if a question might be language-specific; feel free to draw in experience from other languages. A company hires good developers, not specifically Rails, Django, or JavaScript developers. These questions intend to gauge whether you've been involved in the design of a system or feature, or interacted with modules that haven't been taught in a tutorial before, and so forth. Actually getting the question right is simply a bonus.

The best preparation for these questions is to build different things, going back to our earlier section on side projects. While you might not have an exact solution for what they ask, you might have an interesting solution to something similar that you've used previously.

Red Flag: Dishonesty

Never lie about what you know. This guide talks about how to gain a natural edge over other candidates, but I would never encourage outright dishonesty.

As an interviewer, it's pretty easy to spot inconsistencies or an applicant's sense of uncertainty. If you attempt to answer a question by hand-waving over a lot of concepts, it's likely the interviewer will ask you to dive deeper anyway, at which point there's no use in digging an even deeper hole.

Green Flag: Be honest and eager.

Take each interview as an opportunity to learn and to improve for your next interview. If you're stumped on a question, be honest about not knowing, yet be eager to learn.

Be genuinely interested in figuring out the answer as well. It's demoralizing for me, as the interviewer, to explain a concept that the candidate didn't know, only to feel as if the candidate didn't really care to hear the answer in the first place. Being eager to learn helps set the right tone for when you're actually employed and adopting this attitude will make you a very effective programmer anyway.

It's also an old trick for an interviewer to ask something they expect you to not know, either to get a sense of what your limits are or to see if you'll be honest about not knowing.

Red Flag: Not Asking Questions

This goes back to company culture fit. When given the opportunity to ask the interviewer questions, take it. Not asking questions can be construed as the same as saying, "I don't care where I work. I just need a job."

Since research was such a fundamental part of the job application process, you probably won't need to ask basic questions that would have easily been answered within the first two seconds of landing on their site. Asking, “So… what does your company do again?” is a pretty big red flag.

More than just showing a bit of curiosity, asking questions is important for *you*. While it feels like you would be happy taking just any job, you should be trying to work at a *great* job. Remember my story about my first job and how I was immediately unhappy? It showed in my work and everyday well-being, and I wouldn't wish that upon anyone.

Green Flag: Ask genuine, thoughtful questions.

You’ll typically have time for only a few questions at the end of an interview, but here are some examples. Let’s start with some basic questions that answer what you should know about any potential employer:

- What is the size of the engineering team?
- What is your development process like?
- What is your tech stack? Why did you choose to use those tools over others?
- What is the management structure like?
- Do you have a formalized junior mentoring process?
- Do the employees here typically interact with each other after work hours?
- What kinds of projects are you currently working on?
- What kinds of challenges are facing the engineering team?

- What are work hours like for most people? What does *your* day look like?
- If hired, what tasks would I be working on in the first 3 months?
- Is there a performance review schedule in place?
- I've read that the company is currently focused on solving X. Can you tell me more about that?

Here are some meaningful questions that will really drive a conversation:

- What are some things you'd change about the code base if given unlimited time?
- What is something that you dislike about your culture?
- What made you initially decide to work here?
- What did you think of the product/service before you joined and what do you think about it now?
- What is the ideal junior developer to you?

When you land a job, you're going to spend more than half your day thinking about work, talking about work, or actually working. Ask your interviewer questions so that you can get an understanding of how you'd enjoy working at their company. It'll also help if you get multiple offers and need to choose between them.

You can develop a casual conversation based on your questions. Here's a hypothetical exchange:

Interviewer: "That wraps it up. Do you have any questions for me?"

You: “Sure, to start, what kinds of projects are you currently working on?”

Interviewer: “I usually work on a lot of different small projects at once, but the biggest thing I’m working on right now is an analytics tool to track user stats.”

You: “That sounds interesting. Is that an internal tool for admins? What kind of technology are you using to build that?”

Interviewer: “Yep. It’s mostly calculating its numbers from a Postgres database and I chart data visually using a library called Chart.js.”

You: “Sounds fun. I haven’t used Chart.js for charting before, just Highcharts.”

Interviewer: “Nice, I use Highcharts a lot in my side projects. What kind of data have you charted with it?”

And so on.

Be willing to converse with them casually rather than just offering them a forced questionnaire. Be interested in the nuances of their answers, which will lead you to asking better questions.

You’re always trying to show your interest in the company. You do so starting with your application and your golden tickets, as well as during the interview itself.

Depending on the questions you ask, you can also reveal very important things about the company that you might have never found out, for example the company expects engineers to work sixty hours a week and that’s a huge deal-breaker for you. This is a great opportunity for you to prevent a bad situation in the future.

Red Flag: Negativity

It doesn't work in your favor to complain or be negative about anything. It sounds unlikely but you'd be surprised. I've had candidates complain during the technical interview about the specific tool we used to collaborate on code online.

In your job (and life) you will face a lot of unfamiliar and uncomfortable scenarios. The interviewer wants to know if you can handle these challenges with a smile, since they'll be working on them with you.

Different companies have different code bases, some of which might be outdated and riddled with issues. After all, problems like these are why they're hiring more engineers. Complaining about a part of the tech stack, or the version of a language or framework, will not do you any good. Imagine being a chef and complaining that the restaurant you're interviewing at serves bad food.

It's also a potential red flag to share a very polarizing opinion on something that's otherwise harmless. Applying for a web developer position but absolutely loathe working with JavaScript? I wouldn't mention it (but, of course, don't say that you love it or are proficient in it if you aren't). You might be negatively evaluated on that inflexibility when compared to other candidates, even if your job would have only included working with JavaScript 1 percent of the time (your interviewer may not know what you'll be working on if hired).

Green Flag: Positivity and Gratitude

You never know if the interviewer is on the fence about your candidacy. They might be leaning toward a soft no because you may be slightly below their bar for what they expect out of junior-level proficiency.

For every coding challenge I've failed during an interview, I always responded with a post-interview email, thankful for their time (interviewing people is a serious time-sink), along with a complete implementation of what I originally couldn't execute. Not only does this benefit me personally but it also demonstrates some passion and willingness to improve.

It never goes unappreciated—even if you don't get a reply back—because very few people actually go out of their way to do this. This display is so rare that I know some bootcamp grads who have done this and have even been hired by the company, despite an initial rejection.

If I felt an initial phone-screening interview went particularly well, I'll send an email to the interviewer at the end of the day to thank them for their time. It's a nice gesture, which alone should be reason enough to do it, but it also keeps them aware of your existence. This can help speed up your interview process if they're eager to get you through sooner because they have a good feeling about you.

Red Flag: Blaming Others

Similar to the negativity red flag above, under no circumstances should you throw past coworkers, project teammates, or managers under the bus. You are interviewing with a potential future coworker, so saying things like “Well my last project would have looked a lot nicer, but the person in charge of design was lazy and didn’t do much work” implies that you will have negative things to say about your interviewer if they ever dropped the ball as well.

If asked why you left your last company, avoid blaming past managers and coworkers, your salary, or the company’s performance.

Green Flag: Show off your positive virtues.

For those who have held jobs before, even programming jobs, you might be asked why you are looking for a new career path or opportunity. Offer reasons that align with your positive virtues, such as, “I learned a lot at my last development role, but ultimately I wanted to build software to help the less fortunate, which led me to your nonprofit.” Or, “I’m very product-oriented as a developer, so I’m looking to join a smaller company where I can have a bigger impact on the product.”

Mold your narrative to match the company you’re interviewing for. A lot of developers don’t care where they work and it reflects poorly on them during the interview. You want your future employer to think the role can only be filled by *you*.

6

CHAPTER 6

The Company and You

While the company evaluates you, you should also be evaluating the company. Here are some more tips on preparing for interviews and offers.

Know Your Weaknesses

When applying, be honest with yourself about what your strengths and weaknesses are. No, not just because you might be asked the most clichéd question of all time, but because it'll help you formulate a narrative.

What aspect of programming do you have the most trouble with? Is it because you simply aren't interested in the topic, or do you just have trouble learning it? If you don't have a good handle on computer science fundamentals, maybe you shouldn't apply to computer-science-intensive companies like Google until you do.

Consider going for low-hanging fruit first—roles that fit your exact skill set—even if you're not too ecstatic about the company. Your interests can change and you might actually end up loving the company. More interview experience is always helpful as well.

Know Your Strengths

In almost *every* interview I've been asked to tell the interviewer about my background, even if they already read my résumé. In order to tell a good story, you must build from your past experiences. Find ways to convey how your previous roles as a lawyer, student, jelly-bean counter, or whatever you may have been in a past life could benefit the company in the position for which you're applying. Don't embellish, but use these as insights into how you work and what the company should expect from you as a productive employee.

When describing yourself to a prospective employer, whether in a cover letter or in conversation, fine-tune your message. For example, if you were a lawyer, and

the company you're applying to is a small, tight-knit agency, your experiences in a law firm might have involved working closely with clients and working iteratively to deliver a complete product based on client feedback. You're comfortable working in fast-paced environments on tight deadlines and handling many responsibilities at once to get the job done.

But, if the company is a large corporation, you might want to spin your story differently. Maybe your experiences at a law firm have helped you cultivate a respect for process and compliance. You are detail-oriented and/or are passionate about delivering a product that has been thoroughly scrutinized (implying you write complete unit tests, for example), ensuring long-term maintainability and limited blowback.

Different backgrounds add a great deal of intellectual diversity to a team. And, of course, you may *really* want to stress your background if you're an ex-lawyer applying to a law-tech startup.

Being a lawyer is understandably an uncommon, exclusive position, so let's try something more typical. I worked as a cashier/drink-maker at a fledgling small-town café—a startup of sorts—when I was 17 years old. If I happened to be looking for an entry-level programming job at a tech startup right after that experience, I would talk about how my time making drinks offered many parallels:

- The insistence on delivering a great customer experience
- Effectively communicating with a small team
- Effectively working with the management directly as an early hire

- Measuring and improving on my key performance metrics—drink-making time in this case

These are all worthwhile subjects that a prospective startup employer could be interested in. Remember, you're not just a code monkey; no matter who you are, you can bring something unique to the table. Craft your narrative with that mindset.

Know Your Interests

Regardless of your level of experience, you've only encountered a tiny amount of all that comprises the field of programming. If you haven't already, figure out what your interests are and see if that aligns with where you're applying.

Have an interest in real-time video streaming? They might love you at Twitch.tv. Study web security often? Mention it if you're asked about what you like to work on and see where that takes you; maybe the company is working on patching some serious vulnerabilities at the time. You want to instill the idea that you're constantly learning, even on your own time and that you offer more expertise than what the line items on your résumé let on.

I found early on that my interests in programming involved UI/UX design along with database and back-end performance/scalability. When applying to companies, I tried to demonstrate that I could write efficient back-end code while also being able to design clean, usable sites.

I also knew I wanted to work at a small startup, and my generalist interests are sought after in an organization with minimal engineering resources. I might not be the best designer or the best programmer, but I could contribute to any part of the codebase and handle odd tasks at will.

Your interests will clue you in on what kind of organization you'd thrive at. Your interests also have a profound effect on what your career will look like in the future, because you'll naturally become very familiar with the tech stack your company uses. Luckily, my first jobs used a traditional Ruby on Rails stack—I love working with Rails. I enjoy rapidly prototyping side projects, for which Rails is a great tool.

If I got a job coding in an outdated language/framework or maybe a job working on mobile apps instead, it wouldn't be bad; but, for my interests, I'm incredibly glad that I was able to develop my Rails knowledge at Rails shops. Be careful taking an offer at a company where the primary tool or responsibility you'll have is something you're already groaning over. Examples: Employed for back-end when you want to do front-end or vice versa. Employed for developing Android apps if you want to do iOS and so on.

At the end of the day, you might not know what you dislike, so be open to anything, but understand that the first company you work at has the potential to set the tone for the rest of your career, so make it count.

Don't Be Desperate

We've touched on how it's important to not feel desperate for a job. It directly hurts your chances of getting a job if it causes you to be nervous during the interview and it'll hurt your negotiating position, which we'll discuss later, even if you land an offer.

I also want to stress the importance of evaluating how the company would fit *you*, which is hard to do from a desperate standpoint. It's easy to beat yourself up if you're not performing well in an interview, maybe because you're not

interacting well with the interviewers, or the questions are abstract and probably nonsensical.

Companies are all comprised of an arbitrary grouping of people, and they or their interview process might be flawed. If your interviewers don't offer you much respect, and that's something you value, then they're not a good fit for you. Maybe, while interviewing at the company, something about their culture or workplace irked you—for instance, the open floor plan was loud, and people didn't seem to be taking their jobs seriously. That's a reasonable deal-breaker for a lot of people who require peace and quiet when they work.

The first company I worked at didn't have a formalized junior onboarding process, which, at the time, was something I initially told myself was a deal-breaker. I would have identified this before accepting the job if I weren't desperate. Be curious about the environment you would be working in and ask questions.

Impostor Syndrome

It's easy to get swept up by toxic thoughts when interviewing. I always had a subconscious fear that someone would look at my application and say, "Kevin, you've only been coding for a handful of months and you want us to give you an actual job?" The jig would be up—people would know that I wasn't a real programmer and I'd be laughed out of Silicon Valley.

That never happened though. Far from it. People were very helpful even when it was obvious that I wasn't proficient enough to land a job at their company.

Experienced developers deal with impostor syndrome as well. I've found that the developers who feel this way are actually intelligent, over-qualified people. It

seems counter-intuitive, but I think it's because having the self-awareness to question your skill set and not succumbing to arrogance is a gift.

You don't have to be the best developer or even be better than your peers. Stop comparing because, at the end of the day, you don't have to be the best to generate tremendous value as a developer.

Can you style a webpage using a design mockup? People have collectively been paid billions since the dawn of the Internet doing exactly that. Can you create a microposting platform? Twitter is worth over 30 billion dollars right now. Of course there's more to it than that, but to think you aren't entitled to some miniscule fraction of that enormous pie at your skill level is silly.

You've worked hard to get where you are. Reading this guide shows that you have the innate motivation to improve your situation. You care about your craft. Possessing these traits is proof that any impostor syndrome concerns you may have are completely unfounded.

Negotiation

There are many tactics to negotiating, and becoming a savvy negotiator is an art that can take ages to perfect. Instead of teaching you how to be a better negotiator, I think it's a great head start to let your actions do the talking for you. The combination of being profoundly impressive throughout the interview, being selective about the type of environment you want to work in, and not being desperate will give you a natural leg up and can drive up your initial salary offer, even before you've begun to negotiate.

So much is built into not wanting to risk company resources on a bad hire. If you're able to blow past that fear and show the employer you're not like the

other candidates, the employer no longer sees this as a business transaction but is instead asking herself, “What can I do to make sure this person accepts right away before another company has a chance to hire him/her?”

Everyone deserves respect. You've shown respect to the company by carefully crafting a personalized cover letter, demonstrating genuine passion for the company (golden ticket strategy), and showing that you exemplify the virtues of an ideal junior developer. It'll be hard for the employer not to reciprocate that respect. Their perception of you will be different from that of the typical junior archetype they had in mind, which can positively affect your candidacy and salary.

Imagine if Andrew Kim, who redesigned the Microsoft brand as his golden ticket, ended up receiving the bare-minimum salary offer Microsoft gave to other junior designers. It would be a slap in the face, especially since Andrew had other job offers and his talent was clearly recognizable by many. When a company is excited about you joining, they tend to also offer a higher base salary as extra incentive to lock you in.

There is still much to learn about negotiating, but I prefer to spend time focusing on getting hired by the right company as opposed to getting hired at the right salary. If you’re interested in learning more about developer salary negotiations, the single best resource I can recommend is Patrick McKenzie’s blog post titled Salary Negotiation: Make More Money, Be More Valued.

Multiple Offers, Now What?

Congratulations! Deciding between multiple, feasible offers is something that will come down to personal preference. You may consider factors like growth

opportunities, organization size, or company mission, but most people just decide based on major considerations like compensation, location, and perks.

Unfortunately, most people also think about these the wrong way. Here are the considerations you should keep in mind when evaluating a company:

1. Compensation (base salary, bonuses, and equity)

$80,000 in Denver or Toronto offers a different standard of living than $80,000 in San Francisco and New York, where the cost of living is extraordinarily high. Salaries in San Francisco might be inflated to compensate for the increased cost of living, but sometimes not proportionately enough.

There are also seven US states that do not have state income tax: Alaska, Florida, Nevada, South Dakota, Texas, Washington, and Wyoming.

Las Vegas, Austin, and Seattle are all active hotbeds for software development. As the California state income tax hovers around 9 percent, it's possible for many people that a $70,000 offer in Austin, Texas would provide a much better quality of living than $80,000 in San Francisco, California.

Alternatively, living in a small, dense city like San Francisco also allows many people the ability to live without a car. Not having to make car payments, pay insurance bills, or fill up your gas tank saves a significant amount of money.

In short, don't base your decision on the raw salary number, but consider your own living habits, circumstances, and external factors such as state tax.

2. Location (commute distance)

Your daily commute can seriously impact your personal happiness. For instance, while downtown Palo Alto is gorgeous and rich in developer opportunities, I've

decided to avoid working there because the hour-long train commute from San Francisco would consume too much personal time, affecting my mood.

According to author Dan Buettner in an NPR interview (summarized in an article on Lifehacker), we could be *much* happier cutting our commutes:

> *The top two things we hate the most on a day-to-day basis is, No. 1: housework and No. 2: the daily commute in our cars. In fact, if you can cut an hour long commute each way out of your life, it's the [happiness] equivalent of making up an extra $40,000 a year if you're at the $50- to $60,000 level. Huge ... [So] it's an easy way for us to get happier. Move closer to your place of work.*

Additionally, results from a 2014 McGill University study shows that you're most likely to be happiest if walking, biking, or taking a train to work.

3. Perks and Benefits

An employer offering breakfast, lunch, and dinner five days a week while covering the cost of your health insurance effectively reduces your annual expenses by about $10,000. I personally think the true value in free meals lies in preventing you from having to think about what to eat every single day.

Surface perks like ping pong tables, free beer, and plastic ball pits are essentially negligible. Value perks that align with your personal values instead. You may ask yourself:

- Does this company expect me to work over time, preventing me from getting home to my family?
- Does this company genuinely value equality amongst all demographic groups?

- Will my distaste for drinking alcohol prevent me from fitting in during company events?
- Is the company flexible with schedules or do they insist developers attend 9 AM meetings every morning?
- Does the company invest in proper, effective equipment to nurture a productive environment for developers?
- Does the company offer an education stipend for me to further my learning outside of the job?

The Reality of Stock Options

Besides the base salary offer, your compensation will typically include equity in the form of stock options as well. This topic is a little out of the scope of the underlying theme of this guide, but it's important information that many seem to misunderstand.

Thinking about stock seems like complicated (or boring) accountant work, and you're just a developer who wants to code. Maybe in five years your startup exits and you'll make a huge sum of money. Why worry about it now?

If you're joining a startup instead of a large, publicly traded company, you will likely be issued common incentive stock options (CISO aka “stock options”) to purchase your company shares (what you get when given “equity”). This is different from restricted stock units (RSUs) in that the stock options are the *option* to buy X shares of your company at the strike price that was offered. RSUs are stock that is directly granted as a bonus, not as an incentive, and,

while they may need to vest, you still own those shares and their value without needing to exercise them directly.

For example, when Junior Jeff joins a startup, the company's valuation is estimated at $1,000,000. They offer him the option to purchase 10,000 shares, vesting over a period of 4 years, at $0.10 per share. If the company has 10,000,000 shares outstanding, this is the equivalent to a company offering Jeff “0.1 percent equity” (10,000 shares out of 10 million shares total), which is not an uncommon offering for most early stage startups. When Jeff wants to exercise his fully vested shares, it will cost him $1,000 to do so, but he will receive stock probably worth much more than the original strike price.

For example, four years later, the fictional company is now worth $100,000,000. Since the time Jeff joined, the company is now worth 100 times as much! Jeff exercises his option to purchase his 10,000 shares for $1,000. In reality his 10,000 shares are now worth $100,000. Great deal, right?

Unfortunately for you, rank-and-file engineer, your stock options can backfire on you.

1. The company needs an exit for your options to be worth anything, but the vast majority of companies never reach this point. There are caveats: you can sell your options back to the company or sell it to a third-party investor willing to take on the risk of your shares, but these aren't typically common occurrences.

2. You took a lower salary in exchange for equity and the promise of financial success via an exit: the $99,000 payout ($100,000 - $1,000), when normalized over the four years, only amounts to slightly less than $25,000 per year.

Let's say Jeff's starting salary was $60,000. Is it possible that over the next four years, he could have landed a job with a higher salary? If he instead spent a year being paid $60,000, then joined a company that paid him $100,000 annually, over four years Jeff would have made $360,000, which is $21,000 *more* than he would have made staying with the first company that had an exit (assuming neither company offers any raises over the 4-year period).

3. You leave, or are fired, before an exit is in sight: not many people realize this, but your vested stock options expire by default three months after you are laid off or leave the company, so you only own a part of the company if you directly choose to exercise your options at that point.

Back to Jeff's example, let's say after four years, the company was worth $100,000,000 but instead wouldn't exit any time soon. Jeff leaves the company, and is given the option to exercise his options or let them expire. Jeff joined when the price per share was very low, so he doesn't have a problem coughing up the $1,000 to exercise his options. Those shares are now his, and he will get paid out when his company exits.

Unfortunately, while Jeff could easily pay the $1,000 needed to exercise his options, the IRS might unexpectedly want at least $27,720 in taxes. While Jeff bought shares that only cost $1,000 to acquire, they are technically worth $100,000. In the United States, the value of your shares, even if it's virtual money that can't be liquidated yet, adds income on which your Alternative Minimum Tax (AMT) amount is figured.

In layman's terms, If your AMT income amount is higher than your normal income, you are taxed on the AMT income amount instead. Stock appreciation adds to your AMT amount; so Jeff could pay the initial $1,000 to exercise his

options but end up owing taxes on at least $99,000 of intangible income (subtracting the $1,000 purchase price from the $100,000 current valuation of the stock). This can be a significant financial hit, especially if Jeff has no idea when the company will exit, if ever.

The exact particulars of AMT are convoluted, but it's important to be aware of how it can affect you. Andy Rachleff, a cofounder of Wealthfront, writes in a company blog post that the only times when exercising options early (before an exit) make sense are:

> *a. Early in your tenure if you are a very early employee or*
>
> *b. Once you have a very high degree of confidence your company is going to be a big success and you have some savings you are willing to risk.*

4. Your shares are likely to get diluted anyway: it's highly probable that your 0.1 percent holding of a company will no longer be 0.1 percent by the time the company sells. Companies that sell for massive amounts of money are typically companies with many employees, and paying many employees requires much venture capital funding. Your equity will get diluted as the number of outstanding shares increases, but the amount of shares you are given does not. The exact number can vary on a lot of factors, but to give you an idea, you should be prepared for at least 50 percent dilution from seed stage to exit.

Additionally, venture capitalists structure term sheets so that they might get a liquidation preference—a return that is a multiple of their original investment amount before the common stock is paid out (that's where you come in) in an exit. A realistic scenario for Jeff would *not* be a $100,000 payout in the first place. It'll be significantly lower, perhaps to the point of being almost negligible after dilution, liquidation preferences, and taxes are considered.

How you value your equity will also depend on the company you're at. If you're given paltry amounts of equity at a really early stage startup with no product or funding, you should generally treat it as worthless. At a company worth billions and poised for an IPO, the value of your shares is more clearly defined, and it's probable that you won't need to worry about exercising your options before a liquidation event anyway.

Without getting into specifics, these are my recommendations for people searching for work in startups:

- Avoid accepting more equity (in the form of stock options) as compensation for a lower base salary.
- If given stock, be aware of how many shares the company has issued in total, what the company is aiming for in terms of an exit, when they expect it to happen (though much earlier-stage companies will have no idea usually, as there are bigger battles for them to deal with at the moment), and whether your vesting period is accelerated in an exit.
- Talk to an accountant later on about the tax ramifications of exercising your stock options. Make an informed financial decision, while also weighing the probability of an exit.

While a lot of the above definitely sounds like accountant-speak, there are cases in which people have rejected an offer due to the stock compensation structure: A Quora Answer to “Has anyone declined an offer to work at Snapchat?”

7

CHAPTER 7

Interviews with the Other Side

See what engineering managers of successful tech companies consider when evaluating bootcamp graduates or other junior-level developers.

Dan DeMeyere

Director of Engineering at thredUP

1. What is the current company size at thredUP (2015) and how many students, from any coding bootcamp, has thredUP hired?

23 in Engineering out of 55 total. We have hired six engineers that came from a coding bootcamp.

2. What made those hires in particular stand out from the rest of the applicant pool, either in their applications or their interviews?

We have a very close knit engineering team. Our web engineering team, which consists of 10 engineers (4 of which come from a coding bootcamp), has not had an engineer leave in over 2 years. This is because we work very hard on our culture, on professional development, and on hiring the right people.

When an engineer on our team brings in a referral, they are saying that the applicant should be part of our family. Our first bootcamp candidate was referred by one of our most senior engineers, who met the candidate, Kerrie Yee, at a meet-up. Having a senior engineer who is highly respected on the engineering team say they met someone who they think would make a great junior engineer is more than enough to get an interview with our team.

In my experience, the number one attribute that differentiates applicants from coding bootcamps from other candidates is attitude. Our bootcamp candidates wanted to learn. They wanted to prove they were up to the task. A lot of other junior candidates come in with a “what can you offer me?” attitude, which unfortunately is not the right frame of mind for a junior engineer candidate

because like it or not, junior engineers don't bring a lot of immediate value to the team.

Whether it's an applicant from a coding bootcamp or someone who recently graduated with a Computer Science degree, you're not bringing any new technical experience to the team and a lot of the practical experience you might have probably won't be applicable in a fully scaled production environment. As a result, an applicant's attitude plays a significant role in the interview process.

All of our junior engineers are peer mentored by a more experienced engineer on the team. This mentoring relationship is an investment the team is making. The mentor will have a significant and prolonged dip in productivity as he/she onboards, coaches, and pairs with the junior engineer to put them on a path that will enable them to make an impact on the team.

If the junior engineer doesn't ramp up, then that's a failed investment of engineering resources, which are precious to any start-up. This is why attitude is so important. A junior engineer needs to be humble, focused, and ready to work hard for us to trust that they will pay back that investment through their contributions once they ramp up.

On a side note, Kerrie, our first coding bootcamp hire, ended up referring four more coding bootcamp candidates, three of which were hired. Kerrie and the other three engineers all still work at thredUP.

3. What are the primary motivators for thredUP to hire junior developers specifically?

First, they haven't developed any bad habits yet. For us, testing is very important and we're able to instill the importance of testing and other best practices we care about as a team early on.

Second, junior engineers aren't jaded (yet). When we talk about code review, they aren't haunted by previous companies that turned them off to code review.

Third, no task is beneath them. There is never of shortage of work to be done on an engineering team so having a helping hand, especially one who is willing to do whatever they can to help, is always a welcome sight.

Fourth, junior engineers provide value to their mentor. Often times being a mentor will make that person a better engineer. It's easy to explain how to tackle a problem or project, but it's hard to explain why you should tackle it that way.

For example, I know how to write efficient integration tests and proper unit tests and I know when to use each. When teaching the difference of these to your mentee, you have to explain things like, “What is a double?”, or, “When do you set expectations and when do you spy?”

Engineers who write a lot of tests know how to use these things, but explaining what they are from a Ruby fundamental standpoint can be challenging. Mentoring someone requires teaching the fundamentals from the ground up, which often times means revisiting your fundamentals so you can teach the concepts properly and I'm a firm believer that revisiting fundamentals every year or two is a very beneficial exercise for every engineer. I've read the RSpec book three times now and I learn something new every time.

Lastly, junior engineers are like NFL draft picks in the sense that they have the potential to be the future of your team if you draft (hire) right. Going out and

hiring an intermediate or senior engineer is hard, very expensive, and there's not an easy way for you to know that they will work out on the team.

This is why cultivating a junior engineer's career so that they become a intermediate or senior engineer over time is priceless because all of their experience lies within the company's domain, where they know all the processes inside and out, and they have a vested interest in the company's success.

By the way, I was hired as a junior engineer four and a half years ago. Junior -> Intermediate -> Lead -> Director. I bleed green, our company's color. I would lay down on train tracks for thredUP. They facilitated my career ambitions and I show up every day with a smile on my face working hard to pay them back for what they have given me.

4. What has surprised you most about the quality of those hires once they were on the job?

When we hired a number of engineers that came from coding bootcamps, I was surprised by the way the dev team's dynamic changed. Unlike CS grads, coding bootcamp grads typically have work experience that lies outside of engineering.

We have a teacher, a biochemist, a marketing manager, a SaaS salesman, an IBM consultant, etc. and they all bring something unique to the team. The former marketing manager is very good at gathering requirements because of her previous role and it makes her a huge asset to any project because of her meticulous approach to project management and QA. Our former SaaS salesman is gifted at collaborating and communicating with product and marketing, which enables him to sit in on their meetings and help them work through ideas.

In addition to these unique skills they offer, they also help us evolve our engineering culture in new ways. Typically when we meet for our sprint retrospectives or we do an engineering offsite, we reflect on our culture, our architecture, and our processes.

Having the professional background diversity that our coding bootcamp engineers bring to the table provides the team a number of unique angles on how to approach these topics. There are many non-engineering professions that approach culture or processes in a way that we can learn from and our coding bootcamp engineers enable these insights for the team.

5. Can you talk a little about the interview process at thredUP? What types of qualities or traits do you evaluate during an interview? What kinds of questions and challenges do you give a candidate?

Our interview process varies based on the role a candidate is interviewing for. The qualities or traits we evaluate for a junior engineer is very different from how we would evaluate a senior or lead engineer. Our engineering leadership team (CTO, VP of Engineering, Director of Engineering) maintains an engineering core competency document that breaks down each level (intern, entry, junior, mid, senior, and lead) with what is expected for each level in terms of communication ability, attitude, technical skills, and experience. This document fuels what we're trying to assess during an interview.

For a junior interview, we start off with a phone screen that is typically performed by the hiring manager or the engineer that would be the candidate's mentor if they joined. The purpose of the phone screen is to determine whether the candidate's expectations for the role are aligned with our expectations for

the role and whether they would be a good fit based on their experience and attitude.

After a phone screen, the candidate is brought in for an onsite interview loop to interview with 3-4 engineers from the team they're interviewing for as well the hiring manager. The hiring manager and the interviewers discuss ahead of time what the objectives are. For example, one interviewer might be trying to assess the candidate's experience level for a technology they listed on their resume as one of their strengths. If someone says they're strong at JavaScript, we'll have our best front-end engineer put that to the test.

To people who are being interviewed, my advice is to be honest and humble about what you know. If a candidate leads on that they know something better than they actually do, our interviewers will be able to discover this quickly so it's better to own up to the shortcomings.

Another interviewer will try to assess the cultural fit for the team. Would engineers like to work with this person? Would the candidate work well with our processes and workflows? Is the candidate bringing something unique to the team? Someone else will be tasked with interviewing their knowledge of our product. If you haven't signed up for our service, that's a big red flag. We expect candidates to do their homework before interviews. We also expect them to have thoughtful questions to ask.

Another tip to interviewees: we review the questions and answers during the interview loop. If a candidate says one thing to one interviewer and a different thing to another interview for the same question, we'll know. If the first interviewing group discovers a weakness or strength we want to explore, the third group will be informed to press on that topic.

The hiring manager will probably have their own unique list of qualities he or she is trying to assess. Why did this person make the career switch? Why did they leave their last job? Would their last employer re-hire them if they had the chance? These are questions that are trying to surface red flags. If a red flag is found, the interviewer can dig in. These are typically the same questions that will be asked during a reference check so if a candidate misleads the interviewer, we'll know.

After the loop, which has the highest attrition rate of any phase of the interview process, we'll huddle as a team to discuss next steps. For junior engineers, we typically do a hands-on pairing session next followed by a reference check before an offer is made.

At the end of the day, the team needs to be excited about the candidate. If everyone isn't excited to work with that person, the odds are very low that they will move forward in the interview process.

Leonard Kiyoshi Bogdonoff

Senior Software Engineer at Conde Nast

1. What is the current company size at Conde Nast (2015) and how many students, from any coding bootcamp, has your company hired?

6 person engineering team, within a larger 300+ engineering department. Our company hired six bootcamp students in 2013.

2. What made those hires in particular stand out from the rest of the applicant pool, either in their applications or their interviews?

We created a team to focus on hiring. While in that team, I participated in a few bootcamp portfolio showcase sessions. In the session, four engineers were tasked with identifying the most promising students. We selected students based on their perceived ability to learn. We were less interested in the projects, since everyone had the same projects. Instead, we looked for people who had an attitude that would mesh with our engineering team and smart people with ability to pick up new skills.

We were aware that anything the bootcamps taught would different than our tech stack. As a result, we focused heavily on ability to learn, rather than full comprehension of content.

3. What are the primary motivators for Conde Nast to hire junior developers specifically?

Our company highly values the responsibility of raising new software engineers for the future. Hiring is hard. Finding junior developers who are well positioned to work with existing tech needs is expensive for the company that hires them.

We are in a position to take on the initial cost of raising engineers, for the sake of contributing back to the engineering community.

We could always hire someone who has a solid freelancing client history or poach a engineer from another company. We actively decided not to do this. Instead, we are seeking to foster a culture where young engineers can establish career experience and contribute to the overall community. Conde Nast wants the world to know that they value engineering and they are willing to put their money where their mouth is.

4. What has surprised you most about the quality of those hires once they were on the job?

Initially they don’t know much. The bootcamp we hired from was Rails based. Even though the students are able to push projects to Heroku and use a few front-end libraries, they often don’t know anything outside of their domain. This was a surprise for some of the senior developers. We expected this to one extent, so we designed a program to cycle bootcamp graduates through different departments. We wanted to identify which tech stacks and working environments the graduates would flourish in.

As a large company, we have a diverse set of working environments. Some are highly time oriented and driven by the sales and editorial cycles of magazines. This is exciting and stressful for people who aren't used to it. We also have internal tools and platforms that we develop for the entire company to use. The culture for these teams are a lot more like startups. To figure out where people are best fit, we let them have a run at everything.

5. Can you talk a little about the interview process at Conde Nast? What types of qualities or traits do you evaluate during an interview? What kinds of questions and challenges do you give a candidate?

Even before the interview process, we selected people based on their potential, rather than their experience. We believe in what could be. This is important to understand. In the interview process, we focused on practical technical understanding. We made sure people understood the DOM. We tested for CSS and JS ability. We created small exercises to leverage basic CSS styling and JavaScript events. We cater each interview to the individual who is being interviewed. We start out at a broad area, then drill in until the interviewee can't answer questions.

We try to figure out how the person thinks and communicates. We have some questions that are verbal and discussion oriented. We have other questions that are coding exercises, where the person can use search on Google for answers. The point of all of this is to figure out if the person is comfortable asking for help, capable of communicating their issues, and able to find answers to their questions.

The worst thing junior developers can do on the job is try to solve a problem on their own and waste an entire day. What we prefer to see is someone who is willing to find the person who can answer their question. Based on their comfort of speaking up when they get stuck, we believe anyone new can get up to speed.

Timothy Wong

Founder/CTO at Cardpool (acquired by Blackhawk Network)

1. What was the company size at Cardpool and how many junior developers had Cardpool hired at the time of acquisition?

The engineering/product teams had five engineers, including one designer/front-end person, and not including me and my cofounder. I'd consider 2 of the hires junior developers.

2. What are the primary motivators for Cardpool to hire junior developers specifically?

The biggest reason is that it's hard to find good engineers. For our first engineering hires, we recruited senior engineers, because we wanted to have some experienced devs to help guide the team as it grew. But after those hires we realized that it would be hard to constrain our candidate pool and grow the team quickly enough. We knew that there were parts of our code base that could be reasonable entry points for someone relatively new to Ruby, Rails, and web development so we felt confident that junior devs could ramp up quickly.

A secondary reason is that it's good to have a diverse team. Diversity can and should be considered among many different dimensions, but I'll comment about it in the context of seniority in this case. If you have several senior devs working on things, they may tend to get more siloed. A junior dev may be more likely to seek input from others about what they're doing. In a practical sense, this increased communication helps make sure some people are filled in on what others are working on.

This diversity also helps everybody learn—senior devs can teach junior devs some new things, but sometimes junior devs know about the latest on languages/libraries/frameworks/best practices and can fill in the rest of the team too. It goes both ways :).

A convenient aspect of hiring more junior devs is that, as the team grew, it allowed us to offer some mentoring responsibilities to senior devs as a growth opportunity if that's something they were interested in.

3. Can you talk a little about the interview process at Cardpool? What types of qualities or traits do you evaluate during an interview? What kinds of questions and challenges do you give a candidate?

The interview process mainly had two stages. The first was an screen-sharing/ phone interview where we get a quick calibration of the candidate's coding ability. The questions are designed to level up in difficulty quickly, covering the most basic questions to higher level algorithm questions. The interview could last up to 1 hour.

Answering all the questions correctly isn't a hard requirement. Interviews can be stressful and everybody makes mistakes from time-to-time. The most important thing is that we get a decent gauge of somebody's programming proficiency and can see that, if they made a mistake, there's a good chance they would have found it and have been able to fix it.

The second stage is an in-person interview loop with several team members. This gives the candidate a chance to meet most of the people they'd be working with. Each of the interviews are roughly 1 hour long (breaks allowed of course!) and the interviewers have more involved exercises that generally take most of the time. The questions asked may cover designing a complex algorithm,

advanced web development concepts, or require a significant amount of code to be written. The candidate and interviewer work at a computer so that the candidate doesn't have to go home with dry-erase all over their fingers :).

4. Did the junior developers who were hired exemplify any traits or behaviors that made themselves stand out from other candidates?

If I recall correctly, 80 percent of candidates didn't pass the first-stage interview. So practically speaking, the most important way they stood out was by being able to quickly and efficiently complete basic coding challenges.

The junior developers that we hired showed that they were able and eager to learn new things. If they hadn't had many years in industry, they demonstrated that they had acquired a lot of knowledge in a short amount of time. In Cardpool's experience, this meant that the candidates were able to reason about some advanced web development topics such as performance, scaling and security.

5. What is one piece of advice you would offer a junior developer looking to maximize their chances of joining a fast-paced tech startup?

Apply the Pareto principle to your interview prep. In terms of knowledge/ramp-up, it's worth spending time going through the common topics that you're going to be asked, since they're pretty similar from company to company.

Also, do your research on the company before applying. While our engineering team was small, it was very tight-knit, and we could only build the team we did by hiring people who were interested in helping us solve our problems.

8

CHAPTER 8

Closing Thoughts

Some final things to keep in mind before you begin applying.

Make the Most of Your Time

Since companies only interview between the hours of 9–5, I would spend any time not interviewing (typically at night) doing one of these things:

1. Starting and finishing side projects to improve my abilities across the board and to have projects to talk about during interviews.

2. Working on my Github portfolio, leveraging @yourfirstpr, and keeping up to date with the software development ecosystem by consuming blog posts and other media. I personally love the Five Minutes podcasts produced by Code School, which are short podcast episodes that keep me up to date on recent news and articles in the community. It makes for great conversational fodder during interviews.

3. Identifying prospective employers—companies in my area and/or use my specific language/framework. AngelList is a great tool to narrow your search. You aren't restricted to just junior-level positions, but companies that have openings for a junior position often have more resources available for training and are more likely to hire juniors, for obvious reasons.

4. Researching companies and developing my golden tickets for my favorite companies.

5. Practicing coding challenges on sites like Interview Cake, Project Euler, and Code Wars (my favorite). You may also want to read *Cracking the Coding Interview* for practice with algorithms and data structures.

6. Attending meetups, if there are any in my location.

When I first looked for a developer job, it became apparent that I wasn't just trying to find a job, I was trying to find a way to keep food on my table (well, friend's couch at the time). While I joke a fair bit throughout this guide, finding a job is one of the most stressful things in the world and employment is a serious issue for many people.

To that end, I rarely did anything that didn't propel me closer to a job. I consumed copious amount of resources, talked to mentors, built projects, applied for positions, and interviewed every day. Each day I felt I was getting closer to a job, but each passing day was also very uncomfortable. If it took another month or so to get hired, I'm not sure if I could have stayed in San Francisco, so I never let a day go to waste.

Why Rejections Don't Matter

There are countless reasons why you might not be given an offer, many of which don't even have anything to do with you. These reasons include company politics, bad timing, their funding drying up, and so on. So if you receive a rejection, don't take it personally.

It has been hinted throughout this guide that the interview process is broken. While you might interview with a well-trained developer, she may not be a well-trained *interviewer*.

When you get rejected, it often feels as if it's a universal indictment against your abilities as a developer. You must keep in mind that people get hiring decisions wrong. A lot.

A great example of this is when Max Howell, creator of the package manager Homebrew, applied for an iOS position at Google but was rejected due to his

inability to invert a binary tree during the interview (you can see the rest of his strongly-worded tweet here).

Hiring correctly can be tough. It's entirely subjective and very few people are taught how to interview. Even when decisions are data-driven, heavily scrutinized, and analyzed by very smart individuals, mistakes will be made.

Venture capitalists reject great companies constantly despite spending their entire careers learning how to evaluate opportunities. Brian Chesky, co-founder of Airbnb, posted a handful of rejection letters he received from smart Silicon Valley investors in 2008. Brian wrote that for "$150,000 [these investors] could have bought 10% of Airbnb," which is, at the time of this writing, worth over 20 billion dollars.

Interviews are unpredictable, subjective evaluations, which is why tactics that help you stand out, like a golden ticket, work so well.

So happens if you do get rejected? It doesn't mean you have to burn your bridges with that company forever. As an interviewer, I've had to turn away a lot of excellent people because we unfortunately didn't have the resources to mentor juniors at the time. I have no doubt that, in the six months or year that has since passed, those engineers who were once considered too inexperienced might be great fits today, even as a mid-level developer.

If you make a good impression on a smaller company, they'll remember you if you apply again after a stint at an internship or similar posts. Reach back to them just to get a feel for what their hiring situation is like; you have nothing to lose, and they'll appreciate always being considered.

Embrace your rejections. You have not failed, and you can only improve with each iteration.

The Framework

By now you've been thoroughly introduced to the framework to becoming a better job candidate. The first part was becoming a mind reader: understanding how your future coworkers thought and minimizing any opportunity for them to reject your candidacy. Afterward we discussed how employment is largely about a human fit rather than a technical one and the ways you could be a company's most interesting candidate at any given time.

Taking the few hours to learn these concepts and then applying them over time will give you the foundation to always be a remarkable candidate. Just like how you wouldn't start coding a new project without thinking about your design and implementation, I hope that you now understand a successful job hunt doesn't involve just blindly sending out your résumé.

This framework isn't specific to just developers, nor is it specific to just junior developers. You can use this throughout any career, no matter what the position is or what you’re looking to do.

Landing any dream job requires being a dream candidate and you’re already well on your way.