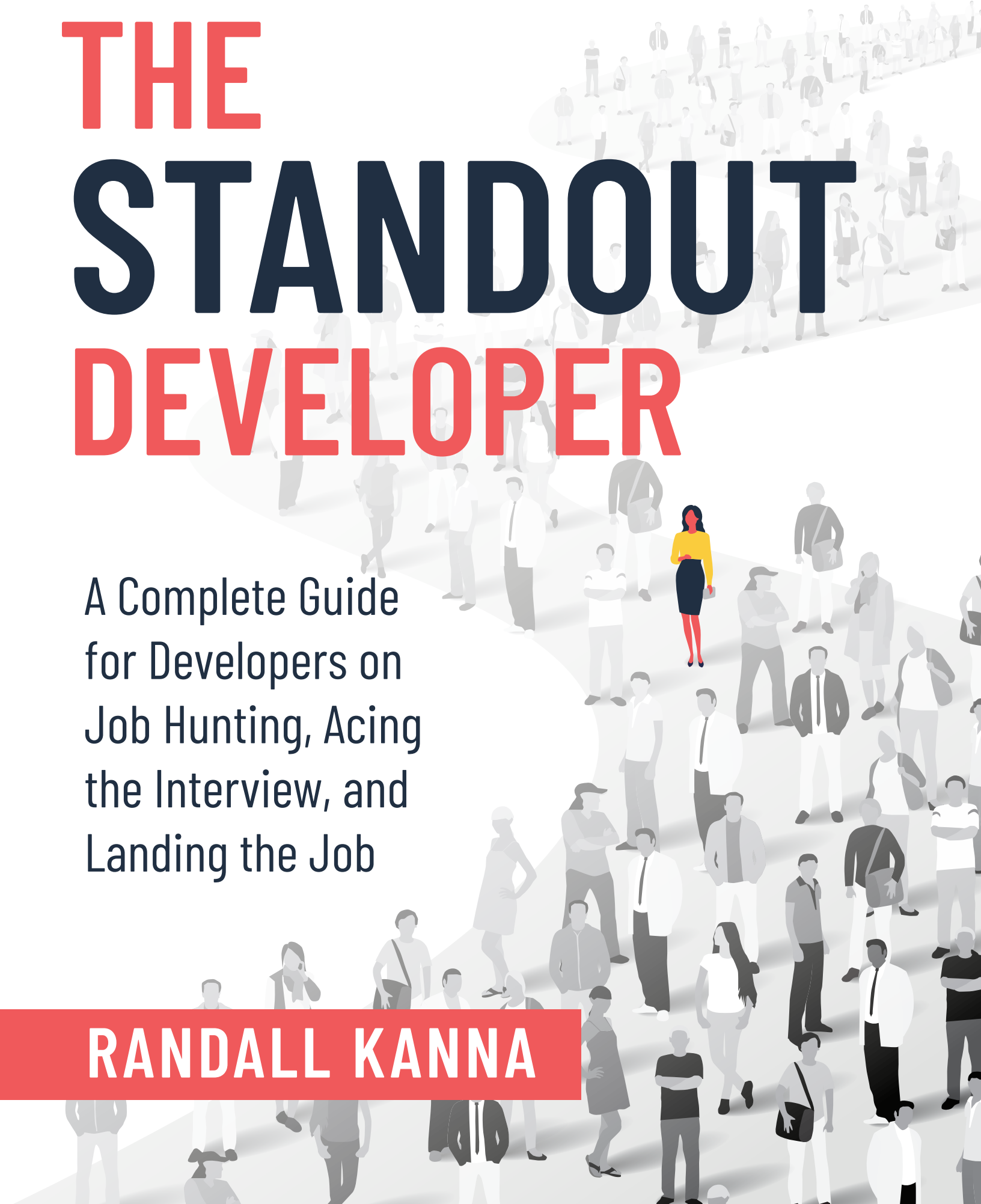
THE STANDOUT DEVELOPER

A Complete Guide for Developers on Job Hunting, Acing the Interview, and Landing the Job



RANDALL KANNA

The Standout Developer

A complete guide for developers on job hunting, acing the interview, and landing the job.

The Standout Developer

Randall Kanna

CHAPTERS

Chapter 1: Standing Out **4**

Chapter 2: Establishing an Online Presence **7**

Define Your Niche 7

Leveraging Twitter 8

Building a Twitter following 9

Junior Engineers 17

LinkedIn 18

Create a Personal Website 22

My TO DO System 25

Chapter 3: Creating a Successful Blog **30**

Write a Compelling Blog Post 32

Creating a Content Calendar 38

Promoting Your Content and Finding Your Audience 38

Best Places to Share Your Blog Posts 39

Free Resources for Creating a Great Blog Post 40

Chapter 4: Landing a Speaking Gig **41**

Getting Ready 43

How to Get Your First Conference Gig 44

Submitting a pitch 45

Tips for Becoming a Confident Speaker 46

Chapter 5: The Standout Resume **48**
The Fundamentals 49
Crafting a powerful summary section 50
Work experience 51
Skills 54
Education 54
Editing Your Resume 56
Common Mistakes That Will Get Your Resume Rejected 57
Final Thoughts on Creating a Standout Resume 63
Craft a compelling story 63
Don't be shy! 63
Showcase your passion 64

Chapter 6: Portfolio **65**
What If You Don't Have a Portfolio? 65
How to Plan Your First Portfolio Project 66
Final Tips to Make Your Portfolio Shine 68

Chapter 7: The Job Hunt **71**
Tailoring Your Resume to the Job 71
Getting a Job Quickly 73
The Biggest Job-Hunting Myths 74

Chapter 8: Resources for Landing the Interview **76**
It's About Who You Know 76
Creating Your Pitch 77
The Quick and Easy Guide to Finding a Mentor 79
Meetups 80
Conferences 81
Hackathons 82

Chapter 9: The Interview and Beyond **83**
The Recruiter Call 83
Passing the Technical Phone Screen 85
The Culture Fit Interview 85
The Onsite Interview 86
Get ready for the "do you have any questions for us?" question 87

Chapter 10: Crushing the Whiteboard Interview **89**
Picking the Language 89
Building Your List 90
Your Study Schedule 93
How to Approach a Whiteboarding Problem 95
My Four Hacks to Whiteboarding Interview Prep 98
My Secret for Getting an Offer in Two Weeks 102

Chapter 11: Cracking the Take-Home Coding Challenge **104**
What Do I Do If I Can't Finish the Take-Home? 104
Four Things Companies Look for in a Take-Home Challenge 105
Here Are the Best Ways to Debug Quickly 107
A Standout Take-Home Challenge 108
Eight Things That Will Get Your Take-home Rejected 110

Chapter 12: The Offer **113**
Compensation 113
Negotiating 116
You're not Married until There's a Ring 119

Chapter 13: Conclusion **120**

Chapter 14: Resources **121**
The Best (and free) Resume Resources 121
Comprehensive List of Job Boards 122

CHAPTER 1

STANDING OUT

How do you become a standout developer?

And why do you need to be a standout? In a perfect world, skill and a good work ethic would be all it would take to land a job as a developer. It's a nice thought, but the world does not work that way. You have to position yourself as a standout.

When I graduated from bootcamp and needed a job, a family friend introduced me to someone at the very top level of senior management at Facebook. This person explained that hiring at Facebook was a "meritocracy," meaning they only hired people purely for their skill and talent.

That may have been true back then, but today's reality is very different.

Take a look at the developer job listings on AngelList or Linkedin and you'll see hundreds of developers applying for *one* job. Most if not all of the applicants are skilled and talented.

And for remote jobs you have global competition because applicants can be from *anywhere* in the world.

The job application process is stacked against you: 98% of Fortune 500 companies[1] use applicant tracking systems (ATS) to process job applications, and 75% of those applications[2] are never read by a human being and are rejected.

You need a roadmap and a proven formula to beat the odds.

It wasn't that long ago that I was one of those applicants applying to myriad jobs and never getting any replies.

[1]"Over 98% of Fortune 500 Companies Use Applicant Tracking Systems (ATS)," Jobscan blog, June 20, 2018. https://www.jobscan.co/blog/fortune-500-use-applicant-tracking-systems/

[2]Kerri Anne Renzulli, "75% of resumes are never read by a human—here's how to make sure your resume beats the bots," make it, March 14, 2019. https://www.cnbc.com/2019/02/28/resume-how-yours-can-beat-the-applicant-tracking-system.html

Even after I became a developer, my career was held back: It was a common belief that *real* engineers didn't have personal brands and that promoting yourself was something that "good" or "serious" engineers just didn't do. I eventually realized this common belief was misguided.

Writing blog posts, having a website and tweeting seemed like a waste of time and a distraction, and none of it would help my career.

I was wrong.

I developed my coding skills and gained a ton of experience after graduating from coding bootcamp, but my career changed when I became a "standout" developer.

I went from applying to countless jobs and never getting replies to having Google, Facebook, Apple, PayPal, LinkedIn and other notable companies reach out to me constantly. I cannot keep up with the flood of emails.

Imagine having iconic, big name companies relentlessly contacting you about working for them. This book shows you how to get there.

Today I speak at tech conferences all around the world. I lead workshops and have co-written a book for O'Reilly, the biggest engineering publisher in the world.

I'm going to share how you too can become a standout developer, and

- I'll explain how I did it all: what worked, what didn't work and the insights I gained
- I'll tell you how I built my personal brand, credibility, an online presence and a large network
- You'll learn how to be a standout in the interview process and how to create a standout resume and portfolio

- I'll teach you how to get your application **seen** and the strategies I used to get to the top of the pile

- I'll tell you what I did to get on the radar of hiring managers and recruiters at Google, Facebook, Twitter, and LinkedIn and others

- You'll also gain insights from other engineers who will share their unique experiences: Daniel Vassallo, Kyle Shevlin, Marcos Iglesias, Ben llegbodu, Abbey Rennemeyer, Shawn Wang and Madison Kanna

Becoming a standout developer will move your career forward, increase your income, and make it easier to quickly land a new job.

Simply put, it will provide you with opportunities.

Just as I was finishing this book, the COVID-19 global pandemic hit. The impact has even been felt in the tech industry, with job listings down 20% according to Glassdoor.[3]

The things this book teaches are desirable at any time, but given the pandemic and current chaotic job market, I believe standing out is more important now than ever.

Ready to get started?

[3] J.R. Reed, "Tech companies pull back on hiring, flashing another grim warning sign for the U.S. economy," CNBC, April 23, 2020 https://www.cnbc.com/2020/04/23/tech-companies-pull-back-on-hiring.html

— CHAPTER 2 —

ESTABLISHING AN ONLINE PRESENCE

Some of the smartest engineers I've worked with have neither a brand nor an online presence. They are simply coding geniuses. If you bring them a problem, they'll understand it instantly and will quickly have a solution. They just *get* things. Companies love them because they keep their heads down and produce large amounts of work. But they have a hard time landing new jobs or getting promoted.

I worked hard at improving my coding skills, and that was my only focus for a long time. As I shared in Chapter One, I once thought *serious* engineers didn't have personal brands.

But I was wrong. My career didn't take off until I created a brand and established my credibility. The more my online presence and authority has grown, the greater the opportunities that have come my way.

Don't get me wrong—you do need to be a good developer. You need legitimate skills and real talent. But having an online presence and being known in your engineering niche means more companies will discover you and want to hire you.

It's not the coding geniuses hiding within a company who receive the great job offers from marquee tech companies. It's the developers who've built an authoritative brand for themselves.

Where do you start? First, you should establish your engineering niche.

Define Your Niche

Before I realized the importance of defining a niche, I felt I had no career direction. I'd see friends who were also engineers advancing in their career faster than I was, and wonder what I was doing wrong. Then I figured out that they'd established their own identity and niche within engineering.

Shortly after this insight, Blockchain became the hot thing in engineering. Very few engineers were specializing in Blockchain, so I devoted my nights and weekends to learning about smart contracts and Blockchain.

Once I'd established my expertise and my niche in Blockchain, I landed my first speaking gig. Two book deals followed, as well as opportunities to participate in workshops at meetups, at the largest bootcamp in the world, and at company events.

Look at the Twitter bios of engineers who have careers you admire. What about speaker bios at tech conferences? Most likely all or most of them will have differentiated themselves somehow. They have a niche and have tailored their public profile around that specialty.

Do you have a niche?

Make sure that your resume highlights your experience in that niche.

It's not enough to be skilled in a niche if no one knows you're an expert. Let's look at ways to take that niche and build credibility and visibility.

Leveraging Twitter

Twitter can be an invaluable tool for engineers to advance their career.

I've found countless job opportunities on Twitter and made many new connections. For instance, the last time I was looking for a job, I posted on Twitter asking who was hiring and the tweet went viral. I received countless responses within just a few hours from companies and individuals that were hiring.

I had people I didn't even know offer me referrals and interviews. I was blown away by the response.

Another time I reached out to a meetup organizer and asked if I could speak and pitched them my idea. The organizer immediately accepted my proposal; I was speaking at the meetup a few weeks later. If you want to speak at a conference or you're searching for a new job, conference organizers, hiring managers, and/or recruiters might check your Twitter profile. It certainly wouldn't hurt if your profile showed that you love helping people and that you contribute to the community.

As soon as I had a couple thousand Twitter followers, things started happening in my career. Meetups, conferences, podcasts, etc., and suddenly, far more job offers.

Building a twitter following

I used Twitter for years with next to no followers. I mostly posted photos of my pets, or memes that I thought were funny. I didn't really think about Twitter as a networking tool and as a result I missed out on some really amazing opportunities. Once I started using a Twitter strategy I landed three podcast interviews, gained over 8,000 followers, and spoke at a conference—all in the same month.

Here's how to implement a Twitter strategy:

1. **Provide value in your tweets.**

   During the time that I wrote this book, I gained 8,000 followers in a month. I did it by making sure that every tweet followed this one rule—provide value. I stopped publishing what I thought was funny or what helped promote my blog posts or my other material, and instead I focused on sharing my own experiences that I thought would add value to others.

This tweet was five years in the making. It was a compilation of all of my favorite CS resources that I had spent hours researching and learning from. The tweet received 9,000 likes overnight and I gained 1,500 Twitter followers.

Here's a tweet that brought me hundreds of followers, over 7,000 likes, and almost 2,000 retweets.

Inside the thread, it had tweets like this, where I shared actual resources that had helped me in the past.

Tweets that help people learn will be incredibly valuable. I've reviewed a tremendous number of resumes and I shared some of the common issues I've found with them. The tweet below got over 3,000 likes because the lessons applied to everyone.

If you're new to engineering, you can contribute value. When I was deciding what bootcamp I wanted to attend I scoured the web for any information I could find from people who'd attended the different bootcamps I was considering.

I would read their tweets and blog posts and email them about their experience. If you're sharing your journey and you share what you're learning, other people will want to learn along with you and from you.

Don't tweet to make a tweet quota. I decided I wasn't going to tweet unless I felt it was valuable. When I see someone tweeting a lot and I don't feel the tweets are authentic, I stop following them.
They seem to be simply trying to reach some quota for the day.

I don't tweet if the content doesn't feel valuable or impactful.

If you have trouble thinking of valuable tweets, try this trick that works for me: I look at my blog posts and figure out ways I can condense them into a tweet while making sure it is still valuable.
I determine the key takeaways and share them in a tweet.

2. **Find an audience.**

   It's not enough if you start posting general tweets. You need to start carving out a specific audience that you want to reach. It might be several audiences. I like tweeting about coding, writing, and how people can find developer jobs.

3. **Spend time analyzing your tweets that do well.**

   At the end of each week, I look at the tweets that brought in the most DMs and interactions. Then I brainstorm ideas for more tweets that are similar.

   Don't focus on your follower count or what tweets will bring you the most followers. Don't focus solely on having a large audience just to look cool. Quality of followers matters, and engagement with people in your industry and other like-minded individuals will lead to amazing opportunities.

4. **At the end of your blog posts, link to your Twitter profile.**

   I didn't originally do this: instead I linked to my newsletter. I realized that people wanted to get to know me and see what I was doing before subscribing to my email list. Twitter provides the opportunity to connect in a real-time.

5. Build a strong Twitter profile.

You only have a few seconds to catch someone's attention when they click your profile. Make sure you have a Twitter account with a professional handle (preferably your name) and your photo. This will provide credibility and also help you with networking later on. I've been to conferences and had people come up to me because they knew what I looked like. They recognized me from my Twitter photo.

You also need to create a powerful Twitter bio. Your bio is a calling card of sorts. It's one of the first things that people scrolling on Twitter will see. Create a Twitter bio that says you are a software engineer and share what you tweet about, like "React," "Node," or "Security."

Include companies where you've previously worked.

You can also keep your Twitter profile valuable by deleting old tweets and ones that are no longer congruent with your brand, message, or focus.

When someone scrolls my Twitter profile, they'll see a collection of my best tweets, and hopefully they keep reading.

Create a powerful pinned tweet that draws people in. It's hard to post a powerful tweet that draws people in every day. But if you create a great tweet that you can pin to your profile, it can draw new followers in.

6. Answer your DMs.

I spent a lot of time responding to people over direct messages, but I never thought it would impact my Twitter growth. It turned out that the more people that I helped and engaged with, the bigger my following grew. I gained super followers that would retweet everything that I tweeted.

You can also craft tweets based on what people ask you. The best way to add value is to figure out what people want. If a few people start sending you the same question via DM, create a blog post based on that question and answer.

For every person that messages you, there are so many more out there that might have the same question but who don't reach out. This is an easy way to provide powerful help without having to do a ton of research.

7. Give away free stuff.

Giving stuff away has been very effective for me. One of the free things I gave away was resume reviews. When I posted a tweet offering to review resumes, I had no idea the response that it would get. It took a long time and it took up my nights and weekends for a while, but the response from people was absolutely worth it.

Many people didn't know how to put together a strong resume or what hiring managers were looking for in a candidate. It felt great to help them. Consider the time commitment involved in offering a free service that takes up time before you tweet it!

I also wrote a free guide on how to become a software engineer. It covers all a reader needs to know about getting started and what it's like working as an engineer. The guide was downloaded over 1,000 times in one month. This was another free resource I created because I saw a need and wanted to help others getting started in engineering.

Interview with Daniel Vassallo, founder of Userbase (@dvassallo)

I greatly admire Daniel. He does a fantastic job with his Twitter account. He also has a Twitter course on creating a presence. After I watched his course, using Twitter finally 'clicked' for me. I started looking at Twitter in a completely different way. I used the strategies he outlined and that helped me gain over 8,000 followers in a month.

Q: ***You're always sharing high value tweets. How do you come up with so many great ideas?***

Daniel: "In the beginning I dedicated a few minutes at the end of each workday to reflect on what had happened during the day. Whenever I felt I'd encountered something that could be interesting to other people I contemplated tweeting about it.
I didn't worry much about staying on one topic: I considered everything that could be interesting. Eventually I started to apply a stricter filter and only tweeted about topics on which I felt I had some credibility. But personal anecdotes remain my favorite tweet type.

This habit of sharing interesting things as they happen became even more natural, and nowadays I tend to tweet about things almost as soon as inspiration strikes. I don't need to spend time reflecting anymore. Many times, I get inspiration when explaining something to someone and I realize that my explanation would be interesting to other people.

Sometimes it's simply when I'm trying to reason about something on my own, and I realize that other people would find my conclusions interesting. Sometimes it's when doing or reading something, and so on. If I could summarize my general approach to Twitter it's to share interesting things that I encounter as they happen."

Q: ***What's the one Twitter tip that every engineer should have in their arsenal?***

Daniel: “You're significantly better off as an engineer if other people know about you, and Twitter makes this possible and relatively easy. Obviously Twitter isn't the only way to do that, but I believe it is very likely the medium with the best ROI in terms of effort to exposure.

The attitude to take is to teach what you know, without any expectations. As soon as you learn something that can fit in a 280-character tweet, share it with others. You’ll start building your reputation and credibility at scale, and you get to keep it even if you move jobs or start something for yourself.”

Q: ***How many more opportunities do you have now because of your impressive Twitter presence?***

Daniel: “Having an audience gives you optionality. I had some great success promoting two info products to my audience—I sold $180K in 6 months, almost all organically through Twitter—but apart from that I feel that Twitter is an amazing opportunity machine.

I'm getting an average of 140 comments per day from my Twitter audience, which in and of itself tells me a lot about what people want help with, what they're interested in, want they want to learn more about and so forth. I also got to know and meet several interesting people who I would never have had the opportunity to encounter if it wasn't for Twitter.

I had been on Twitter for years before that and only had a few hundred followers. Mostly my friends! But as soon as I started sharing tweets about job hunting and preparing for the coding interview, my tweets went viral.

Start interacting with other engineers on Twitter and responding to their tweets. There's no point in being on Twitter if you aren't creating new connections and forming bonds. If you aren't interacting with other users and getting interactions from them, your tweets are basically going into the void. I've met a ton of friends on Twitter and was able to get some incredible opportunities. Don't just reply with any statement on someone's tweet. Make sure that you're adding value.

If you're a junior engineer you can write tweets about your coding journey. People want to know what worked and what didn't. When I started out in engineering I loved reading blog posts and tweets from junior developers to see what worked for them on their journey to becoming an engineer."

Junior Engineers

If you're a junior developer and looking for a new job, craft a tweet asking for advice and information. Sharing your job search on Twitter can be a very powerful strategy for creating connections and getting retweets. People want to help you.

If you don't open a Twitter account, I believe that it will be much harder for you to leverage the engineering community on Twitter.

JOB-SEEKING TWEETS

Craft out a tweet that is short, simple and to the point. People scroll past tweets quickly, so you want to make sure your tweet isn't too long.
If you have a required location or you only want remote, make sure you state that.

Here's a job tweet that I created that went viral.

"I'm currently looking for a frontend or full-stack developer position. 5+ years of experience. Remote only. I've worked at companies such as Eventbrite and Pandora. DMs are open. Please retweet for reach!"

List out what kind of job you're interested in, and what technologies you're experienced in. Is the majority of your experience in backend or frontend?

This will help people who are quickly scrolling through Twitter to know if you're a fit for an opening they might have heard about.

Do you have any impressive facts? Include them. I've worked as a frontend, iOS, full-stack and Blockchain developer, so I made sure to add that to my job search tweet. What is something that makes you unique?

After you've created your job hunting tweet, make sure to pin the tweet to your Twitter profile so it's the first thing people see. Every week or so, craft a new tweet that's worded a little differently. Try different approaches to find out what works and what doesn't.

LinkedIn

LinkedIn is your online resume. Most engineers have a LinkedIn account and it's crucial that you do as well. When I was trying to get my first developer job, the company that ended up hiring me was the one that found me through my LinkedIn profile. My profile ranked high in search keywords.

Here's how to make your profile stand out.

Your headline matters

This is the first thing that recruiters see on search. You want to make sure your headline highlights your most impressive accomplishment and your current job title. My bio includes that I'm a published O'Reilly author, as that's the accomplishment that I most want to share with the world.

If you want a job working in React, write frontend developer or react developer in your headline. Keywords matter on LinkedIn search, and having a relevant headline will help you increase your odds of getting found through search. LinkedIn has a feature that will show you how many times you appear in recruiter searches, and you want to optimize that as much as possible.

If you don't have an engineering job you can state, "Looking for frontend opportunities." If you don't feel comfortable adding that, you can change your headline to "Junior Engineer" so you can show up higher on search as well.

Write a compelling summary

When people check your LinkedIn, they're going to scan your headline and photo and then they'll quickly look at your summary. This will be your opportunity to succinctly express a little about yourself and what you're currently looking for in a position.

If you have a job, you can set "Open to new opportunities" on LinkedIn privately so only recruiters will see it.

Use the "Accomplishments" section to your benefit

For developers without a traditional computer science degree, this is the section that can make or break your profile. This will emphasize your passion to recruiters and make your profile stand out.

When I started out as a junior engineer I made sure to add every course I'd taken and every certification I'd earned to my profile. Completing more to be able to add was a good motivator. Make this section as robust as you can and keep updating it.

While I didn't have a CS degree, I did make the dean's list for multiple years in college so I included that. And while I didn't have any work experience as a developer yet, I had spent the last eight months taking online courses and earning certificates.

Think of all your professional wins and all the extracurricular courses you've taken and add them to your profile.

Do you have a portfolio? You can include this under "Project" in the achievements section. Doing this was a huge help in getting companies to email me back when I was still a junior developer with no job experience.

Fill out every section on LinkedIn

If you fill out every section on LinkedIn, not only will you be more visible on search, but you'll also reach "All-star" status. All-star users are 40 times more likely to get contacted via LinkedIn, so it's worth the investment.

You'll need to make sure you have the following filled out to gain All-star status on LinkedIn:

- ☐ Profile photo
- ☐ Experience
- ☐ Skills
- ☐ Education
- ☐ Connections - Try to have at least fifty LinkedIn connections
- ☐ Summary
- ☐ Industry and your current location

Get a custom URL

If you don't have one yet, you're probably using the default LinkedIn URL. You don't want to use that on your resume so it's important to change your URL to your name.

My custom URL is linkedin.com/in/randallkanna. A custom URL will help make your resume look more professional.

Ask someone to write a recommendation for you

This is a great way to build a little credibility. Recommendations show recruiters that you're a real person and can do the job well.

It's not the end of the world if you don't have a lot of recommendations, and it probably won't matter at all later in your career. But if you're a junior engineer, having recommendations can be extremely helpful in getting interviews and a job offer.

Ask a past coworker to write a recommendation for you. Offer to write one in return as well.

Use a professional headshot

A headshot makes your profile much more likely to be viewed on LinkedIn. This is an important part of creating your own personal brand as an engineer.

It's a pain to worry about getting a good headshot taken, but doing so will dramatically increase your chances of getting called in for a job interview.

Don't use a selfie or the same photo you use for Tinder. Ask a friend with some photography skills to take your headshot and Photoshop the background out if needed. Make sure the photo is close up, use a simple background, wear professional clothes, and make sure the lighting is good.

Some meetups even offer days where you can get a professional headshot taken for free. Certain companies and stores also offer days where you can visit and get a free professional headshot.

Create a Personal Website

A personal site is one of the best things you can do to build your engineering brand and one of the most impactful tools for creating a standout career.

A personal site, if done well, can create credibility and showcase your work.

It can also help sell you to employers and help enrich the dev community. There are a ton of benefits to having a personal site, and it doesn't take that long to get one set up.

Unless you have a very unique name, when you're applying to a job it's hard for an employer to find much information about you online. But if you have a personal site, you can link to it on your resume and showcase your best projects and blog posts to recruiters and companies.

Setting up a website can be a simple, straightforward process. I've laid out the steps so you can get closer to building your brand.

Buy your domain

The first thing you need to do is buy your domain name. I love using Google Domains because you can easily set up an email address and Gmail account with the domain you purchase.

My domain is RandallKanna.com, which isn't a common name and my Mom bought it for me a long time ago. Having your name as your domain is ideal but it may not be available, so be creative but professional. I love seeing creative domain names such as BuiltBy${yourName}.com or ${yourNickName}.dev.

Build your site

After you've purchased your own unique domain name, it's time to build your personal site.

If you don't want to spend a ton of time creating a personal site yourself, you can use WordPress or Webflow. Webflow is really cool because you can pick an existing template and easily customize it. You can even create your own template from scratch using Webflow.

If you're up for coding your own personal site from scratch, that's awesome. I love using React to quickly create a personal site.

You don't have to create your personal site from scratch if you don't want to invest the time. In fact, my latest site is built on WordPress for convenience. I also love using Webflow when I want to deploy a site quickly. But it's important to have a few key pages on your personal site.

Make sure your personal site includes the following:

Your personal site should include a blog. A blog is your way to start building your engineering brand and voice. (I'll cover creating your first blog post and more in the next chapter.) Unless you have a very unique name and you're easy to find on Google, it's important that you include your blog posts on your personal site so they're easy for recruiters to find. You can then link to your website on your resume.

1. **Blog**

   Your personal site should include a blog. A blog is your way to start building your engineering brand and voice. (I'll cover creating your first blog post and more in the next chapter.) Unless you have a very unique name and you're easy to find on Google, it's important that you include your blog posts on your personal site so they're easy for recruiters to find. You can then link to your website on your resume.

2. Contact Section

When companies find you through your blog post or Twitter account, you'll need to have a way for them to easily contact you to discuss job offers. You can send them to your LinkedIn or include an email address on your website, or even include a contact form.

3. Projects

Your personal site should house all of your portfolio projects and include a summary for each one so recruiters and companies can see what you've created. Recruiters shouldn't have to search through your GitHub profile to see your best projects.

4. Relevant links

Here are a few examples of links to include:

- GitHub profile
- Any open source repositories you contribute to
- LinkedIn
- Twitter

GET FEEDBACK BEFORE YOU GO LIVE

Before your site goes live, ask a friend (ideally a friend with design experience) to make some suggestions.

Design is not one of my strengths. Whenever I create something for a side project I ask (beg) one of my designer friends to take a look and make some suggestions. If you have the resources, hire a good designer. Building an online presence requires good visuals. A good designer can be priceless.

Deploy!

You can easily deploy a personal site using Google Firebase. I've created a site on React and was able to quickly add my domain to Firebase and deploy my React site. There are a ton of tutorials out there and options for deploying your own site.

If you selected Webflow, it's easy to add your custom domain to your Webflow project and deploy it from there. There are many tutorials online that will help you use a custom domain with Webflow.

An essential part of building an online voice and presence is creating an authoritative blog. Chapter 3 tells you how.

Before we move on to Chapter 3, let's talk about …

Finding the time, systems, the discipline required to do what it takes to be a standout engineer.

My TO DO System

When I first started out in engineering, I thought a senior developer promotion was years, if not a decade, away. So, I didn't set it as a goal because it seemed so far in the future.

Then, about a year into my engineering career, I realized I was already doing the work of a senior engineer; I was helping lead projects, overseeing giant pieces of applications on my own, reworking hiring processes, and helping organize teams. I was contributing as much code as a senior engineer.

So I asked for a promotion. I presented my manager with a comprehensive document on all of my achievements and work at the company to date.

I got the promotion.

Here's the process I created to manage my time, stay disciplined, and get more done to get that promotion. I still do this every day.

My standup updates and productivity changed drastically as soon as I used a note-taking system that helped me to stay organized and focused.

Here's an example of my daily to do document in Evernote.

Thursday, July 18, 2019

Standup
A quick summary of what I did the day before so I can report quickly to my team and also have a record of this later for interviewing.

Meetings
- [ ] Standup
- [ ] Book club

3 Big Daily's
- [ ] 2 hours of book writing. Finish both chapters this week.
- [ ] Start my next big feature
- [ ] Get all my current PR's finished and merged

Daily work
- [ ] Review pull requests
- [ ] Review my notes for tomorrow's 1-1

To Do
- [ ] Walk the dog
- [ ] Eye app

Habits

☐ Gratefuls	☐ 2 hours of writing
☐ Drink 2x water bottle	☐ Print daily ToDo's for tomorrow
☐ 30 minutes of reading	☐ Focused work first
☐ Tidy house for 15 minutes	☐ Bed by 11:30PM
☐ Brush the dogs teeth	☐ Meditate

Standup. At the end of every day, before I shut off my computer for the night, I spend a few minutes writing a summary of all the work I completed that day. I write down each ticket I completed and my big wins of the day. Doing this at standup the next day helps me to not struggle to remember all the things I did, and it helps me to be better prepared for future interviews. You can look through your old notes and see what you worked on and achieved. It is much easier to add these highlights to your resume when you can review all the wins you've documented.

Meetings. Like many many people, I hate meetings. So, I like to check them off one by one throughout the day to gain a sense of accomplishment after having attended them. I take notes on this section if I need to remember something for a particular meeting.

Three Big Daily's. (My most important tasks.) I define the three big tasks that I need to get done each day that will help me move forward in my career and life. These are the important tasks—the ones that will make you feel like you've been productive, even if you get nothing else done for the rest of the day.

Daily Work. This is the mandatory task that you need to do every day for your job.

To Do. My personal task list with all the things I need to check off that day.

Habits. I try hard to improve my habits daily, so keeping a list of them to check off after I complete them keeps my habit streaks on track.

Taking breaks

When you get stuck on a bug, the worst thing you can do is endlessly stare at your screen and force a solution. I've spent hours trying to fix a bug and stayed completely stuck. I'd take a break, go for a walk, or eat something and take my mind off the bug. After returning to my computer I was able to fix it immediately.

I use the Pomodoro technique to stay focused and to remember to take breaks.

When I don't follow the technique, I find myself taking too many breaks and wasting time by aimlessly scrolling Twitter. Or, I don't leave my computer for several hours and my productivity plummets because I haven't taken a break.

Set a timer for 25–45 minutes and stay focused on **one** task at hand. No distractions, do not check social media, texts, etc. After the timer rings, take a break for 5–15 minutes. Start another pomodoro session after your break. I find that I can do about 4–6 pomodoro sessions a day.

How I just 'knew' everything after being told once

At my first engineering job, instead of needing a senior engineer to show me something several times, I started taking detailed notes.

I would copy and paste the exact terminal commands if they showed me how to deploy something, and I would take notes on everything I learned that day.

This process provided me with a comprehensive document of everything I needed to know to quickly solve issues. Instead of needing to ask someone again what a command was or how I could quickly fix an issue with the deployments, I was able to simply search through my own document.

In this document, I include a header with keywords to quickly search through my notes. Sometimes I write: "Remember that you need to change to the QA environment instead of production when running that command." Doing this came in handy many times and avoided creating a production issue.

I've actually found this habit so useful that I've kept it up throughout my career. To this day, I keep an Evernote folder of everything I need to know about my current company.

Don't fake it until you make it. Do this instead.

"Fake it until you make it" is a common saying, and you hear it often in engineering. Pretend to be confident. Pretend you always know the answer.

But here is the problem. You'll never be able to ask questions and expose what you don't know. Faking it until you make it might work for a short while but at some point, people will realize that you don't know everything.

And that's okay.

It's better to realize sooner than later that no one knows it all. Even the most senior engineers and tech leads ask questions. No matter the level of experience, everyone has gaps in their knowledge.

— CHAPTER 3 —

CREATING A SUCCESSFUL BLOG

Having a blog is a big part of having an online presence. In fact, my first tech job found me online through a blog I'd created while attending bootcamp and that I'd continued to publish after having graduated.

Let's look at crafting your first blog post.

Depending on how you feel about putting your thoughts online, your first blog post could be the toughest one to write. But writing your first blog will make the next one—and the one after that—much easier. Even now when I feel like I have writer's block, I commit to writing at least half a page. After I do that, if I've gotten into a good flow I'll keep writing. If not, I walk away and come back to it the next day.

Get into the habit of writing. Once you've broken the barrier and written one post, the posts that follow will be significantly easier.

Interview with Abbey Rennemeyer, editor of freeCodeCamp

Q: Do you think it's useful for engineers to have a blog?

Abbey: "I do think it's useful for engineers to write about what they're doing, what they know, what they're learning, and so on. I think the general consensus in the tech community is that writing about these things is helpful—not only to yourself, but also to others.

Think about it. After you've worked through a tough bug or problem, don't you want to remember what you did? And chances are someone else will have that same problem very soon, so why not document your steps and share them with the community? You're doing a good deed for yourself and for others. What's not to love?

> When you're writing a blog post it's helpful to think about your motivations. Are you doing this so that future you can remember what you did? Are you writing to help other developers? To inspire them? Are you doing it to get your name out there and grow your network? There are many reasons to write, and you should be clear about what yours is.
>
> Along these lines, know your audience. If you want to write an in-depth tutorial explaining how to use x and y tech, that would be a great fit for an educational publication like freeCodeCamp. If you want to write a promotional article for your latest product or release, that's a great fit for your company blog or something similar. Think about who will be reading your articles, and write as if you're speaking to them.
>
> All that being said, don't just write a blog to have a blog or because you think you "should." Yes, anyone can do it (with a little hard work and practice), but that doesn't mean everyone has to do it. If you want to share your ideas, your story, or what you've learned, that's wonderful. Go for it. But don't feel forced."

Ideas for your first post:

- Write about something you're excited about.
- Document your coding journey. Describe what you're learning each week. Create a blog series about it and help other people learn as well. People love reading about someone else's personal story.
- Pick something you learned recently and use the post as a teaching tool. Ask for feedback from your readers.
- Write about a coding bug you solved and how you found the solution. Walk through your thought process and what the debugging process was.
- Create a tutorial on something you taught yourself. Don't worry about the concept being too simple. I've been an engineer for five years and I still Google how to do simple things in Git.

Write a Compelling Blog Post

It's not easy to write a compelling blog post, especially if you're new at blogging. I often struggle to write a good post, but it does get easier and it is worth the effort. A good post can build credibility and help companies find you online when they search for the topic.

I've written many blog posts on hiring and improving the interview process, and because of that, companies want to hire me to help improve their interview process.

Here's my process for writing a compelling post.

First, I ask myself if I'm interested in the topic. Pick one that you're interested in writing and one you believe others will want to read. If you aren't really interested in the topic, that will come through in your post. You'll also struggle to complete it.

Ask yourself whether your post captures the interest of readers. If you happen to read my blog, you'll see that I like helping people by sharing how I learned how to code or got a job in engineering. Most of my blog posts center around these themes because I know my readers like this content.

Once you've settled on a topic, don't change direction. When I first started blogging I had a bad habit: I would start with an idea but convince myself it wasn't good enough, so I'd figure out another topic. This was neither effective nor sustainable. I didn't publish a single blog post for a while because I kept scrapping ideas.

Write every day until the post is done. Set a certain time to write. Even if you just open up the document and stare at what you've already written, commit to that process. The more posts I write, the more ideas I come up with and publish. It can be a barrier to settle on an idea for the first post. Decide on a topic and go create.

Create an outline

Staring at a blank page can be anxiety inducing. If you map out the outline of your blog post before you start, you'll feel as if you have a direction and you'll be able to tackle one section at a time. Having a great outline makes it so much easier to stay on track with your daily writing goal.

Ask yourself what kind of reader you want to attract with this post. What do you want your audience to take away from it? Do you want to compel them to do something? When you're writing the outline, think about what your reader will gain from reading your post. Create your outline based on the questions you've asked yourself.

Example of a blog post outline:

Title: How I got a Job Fourteen Days After my Coding Bootcamp

1. *Treat your job search like it's your job.*
   a. Expand on how it's easy to get burned out if you prepare too much for an interview.
   b. Cover how exhausted I was after bootcamp and how I needed to make sure I stuck to a schedule and kept my momentum.

2. *Make yourself stand out in the interview process.*
   a. Discuss how the reader needs to figure out a way to stand out during the interview.
   b. Share my story of how I applied to dozens of jobs every day and got no results until I customized my resume to the job.

3. *Quickly move on if a company isn't a good fit.*
   a. Share how I only applied to companies that were interested in recent bootcamp grads and those without a CS degree.
   b. Discuss how doing this sped up my job search because I immediately weeded out companies that weren't a fit.

4. *Conclusion*
   a. Summarize the main points.
   b. Share a powerful message so my readers feel they can get a job too!
   c. Add a CTA and include your Twitter handle.

Your outline can also help you understand if you need to do more research before you start writing your post. For instance, in the blog post above I needed to research some statistics on how many resumes aren't even seen by real people.

Create a compelling title

Pick a title that will catch the attention of potential readers. Envision what your audience is searching for and what would catch their eye. If you're trying to attract readers with similar interests as you, think about what types of blog posts you search for and what you click on.

Abbey's suggestions for writing good titles.

Practice, practice, practice. Ha, but really, it's taken me a long time, literally years, to get to the point where I can usually come up with a snappy title. And I don't get it right 100% of the time. Fortunately, Quincy is also very good at this, probably better than I am. Again, practice. I've seen a lot of headlines—probably tens of thousands at this point—and eventually my brain started sorting the effective from the not-so-great.

I usually try to think about ways to grab the reader's attention. Action verbs help, and the names of popular tech stacks/languages work well, like "How I did x" or "The best way to do x" or "How to build x" (you get the idea) often work well.

Headlines should be matter-of-fact—no need to sensationalize or use click bait—and interesting. Is this something you'd want to read based on your headline? freeCodeCamp has a style guide available.

Write the post

After you have your outline, start writing. Often, I find that if I've written the first page, I'm motivated to keep going. If you also hate writing like I did when I started writing tech blogs, commit to writing a certain word count every day or set a required period of time.

Go into detail. Expand on every point. Your blog post shouldn't be too short or lacking detail. When you're writing your post, ask yourself who will be reading it. You need to think about the tone and adjust it for your audience. Are you teaching a subject to junior engineers? If it's a technical tutorial, you need to make sure the post is thoroughly detailed, with step-by-step instructions.

Write a powerful introduction

First impressions matter. After you've created a compelling headline that captures readers, you want to make sure you keep them. You need to create a compelling introduction that grabs the reader and makes them want to read on.

Start with the most interesting aspect of the post. What will draw the reader in and make them keep reading? From now on, when you read blog posts, take a look at sentences that stand out to you and create a compelling introduction to the reader.

Don't forget the conclusion

The post needs a compelling conclusion. Summarize the key takeaways and write a conclusion that inspires your reader to take some action.

What do you want them to do? Should they subscribe to your email list and/or follow you on Twitter? Create a boilerplate CTA for all posts.

Editing

I love running my work through Grammarly. Grammarly is a service that will check what you've written for any issues. Best of all, there's a free version!

You can also ask for feedback from peers who can write. They might be able to find sections you can expand upon, or identify where you might have missed a transition to your next point. If you have the budget, hire an editor. Good editors are invaluable.

Select a good image

Your blog post should include a relevant image. There are plenty of places where you can find free images. I love to use Unsplash. Be sure to give a photo credit.

Decide where you'll publish your post

Medium, freeCodeCamp, dev.to, and your own site are only a few options of places where you can publish your blog post. Unless there is a restriction copyright issue from an outside site, publish the post on your blog and push it on Twitter also. Each has its own advantages and disadvantages. I like to publish my posts to freeCodeCamp because that reaches the most people.

Abbey's suggestions for creating a good blog post.

There are a couple of things that immediately come to mind.

1. Do your research, and don't be afraid to go into detail. Provide data, charts, graphs, code snippets. Link to resources and cite your sources whenever necessary. In-depth tutorials are so helpful—people can bookmark them for later and use them as a resource when they're working through a project. And run your code. Test your links. Check your images. Anything that moves should get a once over.

2. Use a conversational tone whenever you can. I often tell people to explain it like they're talking to a friend or trusted colleague. That way it's easier to follow along and it's easier for the author to tell if their explanation makes sense when they read it back over again.

3. On that note ... proofread. You shouldn't be submitting articles to a publication (or publishing them yourself) that contain typos. This also gives you the chance to hear the article in your head, make sure it flows, and reword bits as needed.

4. You want to help the reader get through your article as quickly and easily as possible. Anything you can use to accomplish this—use headings/subheadings, write a brief but informative introduction and conclusion, make sure there are connections between sections—really makes a difference. Again, think of that conversation and the flow that usually comes with it.

5. This may sound silly, but try to make it interesting and engaging! Some of the most entertaining articles I've ever read are about (relatively) pretty dry topics like this one. I know it's not easy, and it probably comes more naturally to some people than others, but it's okay to be funny. It's okay to write playfully at times. If you're writing for a tech publication you want to be serious and authoritative and all that, but try not to make it so dry that no one actually wants to read it.

   (I listed this point last, as it's not critical, and it's really not always possible. It's just a bonus that makes articles extra fun to read.)

Finally … **post it!**

Creating a Content Calendar

A few years ago, I was only posting a few blogs a year. I didn't have many readers and it felt as if my posts were just going out into the void, never to be seen again.

The only time I've been able to stick to a schedule and regularly post has been when I've had a content calendar. A calendar can help you keep track of what you want to write about and when it should be published.

I keep a whiteboard calendar that has a notes section that I use for Post-It notes with content ideas. Every time I come up with a potential new post idea, I write it on a Post-It note. Once I've decided on a new idea, I move the Post-It onto the calendar and pick a date that I want to be ready to publish.

I've found that when I have a set date for a post to be published or submitted to a site, I'm more motivated about meeting the deadline. And there's something satisfying about crumpling up the Post-It and tossing it in the trash when I'm done.

Create your own content calendar. It doesn't have to be a white board with Post-Its. It can be whatever you want. But having the set accountability to get your blog posts done will make a huge difference in the amount of content you'll create.

Can you commit to one blog post a month to start?

Promoting Your Content and Finding Your Audience

Building a list of readers will motivate you to write more blog posts. When my posts were going out into the void and I didn't know I had readers that were excited about reading more, I wasn't motivated to write posts.

Once you've written your blog post, you can't leave it at that and expect to get readers. You need to actively promote your post and find people who are interested in the content you're creating. You can't post your blog posts on your personal site and expect readers to find them. You need to share your content where people can find it.

Best Places to Share Your Blog Posts

After you've hit Publish on your blog post, you need to drive traffic to your blog. If you write something and you don't find a way to get readers, it's definitely just going out into the void.

Here are a few places you can post your blog to get some traction:

- Twitter is my favorite place to share a new blog post. Of course, I had to build up an audience there first by creating valuable content and helping people.

- Reddit is another option for posting your blog. Find a suitable subreddit and try to create a catchy headline for your post. I had one of my blog posts trending for a few days on r/javascript, which brought a ton of traffic to my blog post.

- FreeCodeCamp is a great place to post and receive thousands of readers. One of my blog posts on freeCodeCamp received 30,000 views in one month. You will need to be accepted as a freeCodeCamp author before you can post.

- Hacker Noon will let you immediately sign up to post your blog post there. I'm always aimlessly scrolling through Hacker Noon to find new posts, so this is truly a great option to drive traffic.

- Hacker News is another option but I generally don't recommend it for most blog posts. You tend to have a huge amount of competition on there and it's hard to get any readers.

- Dev.to is another great place to post a blog, but you'll probably need to post it on other places I've listed above to gain serious traffic.

Free Resources for Creating a Great Blog Post

- Free online editing tool that I use on all my posts
- A post from Seth Godin about building an audience
- I love using Unsplash, but there are a ton of places online where you can find free images as long as you give the photographer credit.
- Canva, a free graphic design tool
- Google analytics to find out who is reading your blog posts
- Loom for great desktop recordings
- I've used Audacity for ages. It's a free audio recorder and audio editor.
- I use ConvertKit to build my email list. It's easy and simple to use.

CHAPTER 4

LANDING A SPEAKING GIG

Speaking at conferences is one of the most valuable things you can do to build credibility as an engineer. It is not as hard as you think to be a conference speaker. When I first started attending meetups and conferences, I would listen closely to all the speakers and think: "Wow, I don't understand half of what they say. I'll NEVER be able to speak at a conference."

I thought speaking at a conference was something only for the elite tech gods. I was terrified of speaking even at meetups and turned down several opportunities.

In time, I overcame my fears and built credibility by speaking at conferences, which created amazing career opportunities.

Here is how I did it:

I was interested in moving into Blockchain so, in my spare time, I took online courses in Blockchain and smart contracts and eventually learned enough to teach a Blockchain workshop. (I touched on this in the section about defining your niche in Chapter 2, but I'll go into more depth here.)

And I continued to share on Twitter what I was learning in Blockchain.

I then applied to speak at a Blockchain conference, using my background in Frontend, and landed my first Blockchain speaking gig. I tweeted the details on the conference and my speaking event.

After the conference, a prominent tech publishing company reached out to me because they watched my talk and now considered me an expert. That conversation led to another opportunity with another publisher—O'Reilly, the largest tech publisher in the world. I signed a deal to co-author a book on Blockchain and smart contracts.

I suddenly had a new job and a book deal to boot.

Interview with Marcos Iglesias, senior frontend engineer with Lyft

Q: What tips would you share with speakers who get nervous on stage?

Marcos: "First of all, I'd like to make clear that everybody gets nervous on stage! Even the most experienced speakers do. Some speakers do it just for that feeling, for experiencing that thrill! That being said, what helps me is using a couple of thought "reframings."

The first one is telling myself, "I am not getting nervous, I'm just getting excited about it!" This is not a lie ... being nervous and excited are basically the same, it's just that the former gets bad press.

The second reframing I do is related to the reason I'm speaking. I avoid thinking of myself as an expert or that I'm going to teach the audience how to do things the right way. Rather, I see myself as a humble engineer who learned something helpful along the way and who wants to **share it with others** to save them some pain, or just to be useful. Why would I be nervous then? I just want to help!"

Getting Ready

Here are some great ways to build up to speaking at a conference.

Speak at a meetup

I wanted to start speaking at engineering events but I was filled with fear at the idea. What worked for me to overcome my fear of speaking was to accept a meetup offer to speak in public. A meetup was the best venue for me to get started. Meetups are typically smaller, informal, and in more intimate settings. They are less threatening and you might even know many of the people in attendance. Meetups can have lightning talks, which are a perfect place for you to start. A meetup will generally have several lightning talks that are just a few minutes long and two to three main speaker events. (See below about attending meetups that you are interested in speaking at before you arrange your first talk.)

I asked a meetup host if I could do a talk. They immediately said yes. It was that simple. So I can say to you: Get over yourself and pitch! There's no tech troll under the career advancement bridge heckling you or barring you from speaking at meetups. I've since learned that organizers have a hard time finding speakers to fill spots. You might do the host a favor by volunteering.

One tip is attend meetups you're interested in regularly and offer to help out. Meet the attendees. Get comfortable with the group.

Another is to build a relationship with meetup hosts. When you are ready, talk to the meetup hosts in person or pitch your idea for a talk via Twitter.

Create YouTube tutorials

If you don't have a list of past speaking gigs or recordings of the talks, create some online tutorials that conference organizers can view.

A video series is an even better idea. This helps conference organizers get a feel for how you present on stage.

Pick a concept you know well or even one you want to become more knowledgeable about and film a tutorial. Plan out what you want to cover and then create a slide deck. Download a screen sharing program and record your screen.

Post this online and include it when you apply to speak at a conference.

Video tutorials are not as valuable as previous speaking experience, but they will help you stand out.

Blog on a platform like freeCodeCamp or dev.to

Another great way to build credibility is to write. Publishing on a platform like freeCodeCamp, in particular, can help build your reputation as a reliable and valuable member of the community.

Blogging on a well-known platform will help improve your Google search results. When the conference organizer googles you, they'll find your content and you'll be more credible.

How to Get Your First Conference Gig

Interview with Marcos Iglesias

> ***Q: What's the one thing someone should know before submitting their first talk proposal?***
>
> **Marcos:** "Nothing, just do it! At the same time, many things. First, know that the acceptance ratio is really low! Don't feel bad if you don't get selected. It's better to send your proposals to a good bunch of conferences and then forget about them. What a surprise when you get selected for one!

> The second is that getting selected is only the beginning. If the organizers pick your talk there are still many things to figure out, like if their budget covers your expenses (or if your company can take them over). Maybe the conference date was suitable at the beginning but now you have other commitments. It is all a process, and it's better to just take it easy.
>
> Another thing you should know is that there are many resources on the internet about preparing good conference proposals. You should read and follow them. They are useful, as they teach how to create slick and concise writing, which is a skill that's very worth acquiring."

Learn about the next big thing

Blockchain or machine learning are hot topics and organizers and conference organizers are looking for speakers. If you are interested in the next "hot" thing as I was in Blockchain, it is easier to stand out from the crowd as the competition is smaller. You don't need a robust speaking resume if you can give talks in an area that few are knowledgeable about.

Submitting a pitch

First, you'll need to make a list of conferences. You can search online and find a list of conferences and the deadlines for speaker applications. Create a list of conferences that are feasible for you to speak at and prioritize in terms of application deadlines.

Next, start brainstorming a list of conference topics for your pitch. You'll need to come up with two or three topics for your speaker application.

Pitch!

Marcos Iglesias

Q: How do you decide on a conference topic to submit?

Marcos: "If I am working on something at the moment that has the potential to become a talk (hint: most of the things have it, it is a matter of finding the right angle), I might write a proposal and send it. Most of the time, I just send my talk portfolio to each conference I want to attend and let them decide.

You could ask, all my portfolio? Yes! The first year I started speaking at conferences, I had only one talk. I set the goal to create two or three new talks every year, so the next year, I had four, the following I had six, and so on. Don't get fooled, even the most popular conference speakers don't reach a 5% acceptance ratio for their submissions. That's why it is essential to send several proposals."

Tips for Becoming a Confident Speaker

Some people are natural speakers. I'm unfortunately I'm not one of them, but I trained myself to be a confident speaker.

Here are a few of my tips for making yourself appear like a natural speaker.

1. **Talk slooooooooooowly.**

   When we get nervous, we tend to speak fast.

2. **Practice constantly.**

   Practice your talk/presentation in front of everyone you know. And I mean everyone. Your mom or grandma might not be able to give you tech tips, but they can tell you that you need to stop saying "Um" every other sentence. (True story)

3. Act natural.

Even if you're shaking and feel sick, slow your breathing and think about what you're going to say next. If your hands are shaking, don't gesture.

4. Make eye contact.

At my first talk, I was too scared to even look at the audience. But do look at your audience. When I did look up, I saw many supportive faces and smiles, which was so encouraging and built my confidence. Good people want you to succeed.

5. Dress up.

At first, I used to want to fit in when I spoke at a conference, and in tech that means casual dress. Later, I wanted to stand out and appear more professional so I dressed up not down.

CHAPTER 5
THE STANDOUT RESUME

In Chapter 1, I shared how the job application process is stacked against you.

I know this because I was on hiring teams at Eventbrite and Pandora for several years and have reviewed hundreds (maybe even thousands) of resumes. I know what hiring managers and recruiters are looking for in a resume, and I know what will send an applicant's resume straight to the "no" pile.

A standout developer needs a standout resume. In this chapter, I'm going to go into detail about what hiring managers and recruiters look for in an engineering resume and how you can ensure yours makes it out of the general pile. I'll show you how to create an engineering resume that gets a reply.

Resume screening

Your resume needs to make it through several screening processes before you get on the phone with a recruiter. There are ways that you can avoid these processes and get your resume into the hands of a real person, and I'll discuss those later. First, your resume will have to beat ATS, which I introduced in chapter 1.

What is ATS?

ATS is an applicant tracking system that scans your resume for keywords and filters that the company is looking for. Many resumes (up to 75%) that are submitted will never be seen by a human being because the resume won't rank high enough after the ATS scan.

If your application includes the keywords that the company is seeking, you'll get through to the next stage. This isn't as hard as it sounds. When applying for a job, just make sure to use the keywords listed in the job posting. I'll discuss this more later.

The CV screen

If your resume makes it through the ATS scan, your resume will next be reviewed by a recruiter or by an independent team that the company hires to review resumes. This recruiter (or team) will decide if you'll make it to the next step of the process, which is a phone screen with the recruiter. Again, having an outstanding resume determines whether you'll get to a CV screen.

The Fundamentals

Your resume doesn't have to be flashy or use a fancy template, but it does need to be formatted well and it should clearly showcase your work experience.

In this section, I'm going to walk you through each step needed to build a great resume.

Using the best template

Select a resume template that is clean and professional. There are free tools like Canva that have thousands of easy-to-customize resume templates.

Here's a site that lists other potential free resume design tools you can use.

Find a resume template that matches your experience level. When you're selecting a template, it's crucial that you select one that's appropriate for your experience level. For instance, if you are a junior developer or you are moving into engineering from another field, focus on resume templates that have sections for personal projects, certifications, other work-related experience, and courses. You'll still want to focus on your past work experience and relate how that experience will make you even stronger in a developer position.

If you're a mid-level senior developer, you have a little more flexibility in the resume template you can select. Depending on your past experience and education, you might want to select a different template to highlight certain things. If you're a self-taught engineer with no degree, you'll want to find a resume that highlights work experience and projects. If you have a CS degree, you should find a resume that highlights the education section.

Crafting a powerful summary section

The traditional, boring summary has become outdated: "Objective: Obtain a goal in Software engineering at a tech company."

Having a powerful summary section can mean you get noticed and stand out in a pile of resumes. More than that, your summary should immediately draw the recruiter in and make them want to keep reading. Show the hiring manager or recruiter who you are and highlight an achievement.

The key to writing a standout summary is to do it **last.** Write your resume first and then, once you've got it airtight, focus on your summary section. Look for the most impressive details in your resume and articulate what you delivered.

The recruiter won't really care that you are an avid gardener or reader or that you love to hike; they want to know that you'll create results for their company ... period.

I recommend tailoring your personal summary section to the job you are applying for, if possible. For instance, if I were applying for an iOS dev job, I would not highlight that I spent the last few years as a frontend developer. Instead, I would include that I taught myself Swift and Objective-C a few years ago. And I did it in a short period of time to step into a role my company needed.

Finally, make sure your summary section isn't too short or too long.

Work experience

Work experience is the most important part of your resume. Recruiters will tend to spend the most time reading this section, so you want to ensure it's compelling. If you only have a few hours to spend on your resume, you should spend the majority of your time working on this section. This section shouldn't focus on bland tasks or contributions.

A hiring manager wants to know *exactly* what you can do for their company. Help them determine that through your work experience section.

Interview with Ben Ilegbodu, principal engineer at StitchFix

Q: ***What makes a resume you're vetting standout when you're reviewing it prior to a phone screen?***

Ben: "My resume vetting mainly determines whether I want to spend 30 minutes on a phone screen. So that vetting mainly looks at whether their resume shows that they have the necessary experience for the role. Resumes that are well designed definitely stand out, but it doesn't have to be something designed in Photoshop. It's just that the content is aesthetically laid out in a way that allows me to get to the information that I want: previous roles, companies, length of time at the company, and responsibilities in the role.

Honestly, what makes a resume stand out the most is relevant experience. There are lots of things candidates with minimal experience can do to stand out from the pack, but the greatest is experience."

When crafting the work experience section, avoid non-specific statements such as:

- Led a project team for the new checkout experience
- Created a web application built in mean stack
- Provided support for the company's main application
- Test driven development using Jasmine
- Practiced object-oriented design patterns
- Created many reusable components

Instead, try to focus on specific results and achievements. Think about the wins and accomplishments you had at each job. Companies want to see exact numbers and percentages from outcomes and projects.

Here is a list to help get you started when noting your achievements:

- If you have released a project ahead of schedule, list that on your resume.

- Describe how you built a product independently and how it was built in whatever tech stack you used.

- Consider how many customers used the product when you launched. If the number is significant, be sure to include this information.

- Did you work on a task that helped speed up the company's app?

- Have you ever saved your company money? An engineer I know saved his company a million dollars. That's something he should list on his resume.

- Have you ever sped up development time for a product? For instance, did you create a deployment tool that let engineers deploy in a few seconds instead of a half hour? List it.

If you don't have any work experience as a developer, that's okay—you can instead focus on expanding your resume projects section and adding extra courses you've taken to the education section. (Take a look at the section later in this chapter that includes my first engineering resume.)

Here are a few examples for your work experience section:

- Independently delivered the "XYZ" project built in Node.js and Vue.js three weeks ahead of schedule.

- "XYZ" was used by 2,500 users in the first month.

- Implemented "XYZ," which improved loading time of ${whatever you sped up!} by ${loadTime}.

- Researched and found an existing automated testing solution that allowed the development team to catch ${bugCount} issues in QA once implemented.

- Created a deploy tool using ${techUsed} that reduced the overall deploy time from thirty minutes to two minutes.

Projects

Adding a personal project to a resume is the best thing you can do when you have little or no job experience. As a senior engineer, I still list projects I'm involved with. Interviewers love to see that you're excited about coding.

Once again, you can start small. Add a project that you completed as part of a coding challenge. The next time you complete a take-home coding challenge, make it great, and use it in your portfolio section (see chapter 6).

Skills

The rule of thumb for a skills section is to only list the skills you feel confident about when answering questions. For instance, I used to work as an Ember developer a few years ago, but I haven't kept up with recent framework changes. Even though I have written entire applications in Ember, I don't feel confident that I'm up-to-date on Ember.

Don't list percentages on your skills in your resume. If you only feel 20% confident in a skill, it doesn't belong on your resume.

Don't list too many skills on your resume. If you have forty different skills listed, interviewers may think you're lying, inflating or overconfident in those areas.

Education

If you don't have a CS degree it won't prevent you from advancing, as long as you position your skills and achievements correctly. In the education section, list the courses you've taken or the certifications you've completed. If you have a degree in another field, you should list it.

If you have several degrees, list the most relevant and recent one first. And if you have a college degree, don't include your high school diploma.

If you are still in school, list where you are. And if you dropped out of college, but only had a few classes left to graduate, I suggest you list that.

Courses

I have over five years of experience working as a software engineer, and I still have a section on my resume that lists the courses I've taken. Even if you are a senior engineer, companies and recruiters still want to see that you prioritize learning and development.

The courses you choose to showcase on your resume don't have to be the Harvard CS50 course. You don't need to only list complicated engineering courses. List what you've done to advance your skills. Start a new course today and put 'In progress' on your resume until you finish the course.

Did you attend a conference and complete one of the training sessions? List that in your education section.

Certifications

Certifications add strength to your resume.

Similar to your list of courses, start simple. If you've taken a freeCodeCamp course, add that under your certifications section.

All of these things will prove to a hiring manager that you have a growth mindset and are willing to continue improving your coding skills and knowledge. As you advance in your career, you'll be able to add more advanced certifications.

Contact me

Make sure the following is included in your contact section:

- Your personal website, which showcases any blog posts and projects you've created
- Your email address (believe it or not, I'm seen resumes without email addresses)
- Your phone number (Be sure to remove this if you post your resume somewhere publicly online.)
- Your Twitter profile
- Your GitHub profile

Once the writing is done, the hardest part is over. But before you start sending it out to potential employers, make sure your resume is properly edited and formatted.

Editing Your Resume

Mistakes on a resume stick out like a sore thumb and make your resume look unprofessional. If you have a budget, hire an editor. Or, you can do what a friend of mine did. She had twenty or more people edit her resume. She spent months perfecting it. I would catch her randomly looking at it again for edits and improvements. I wondered why she spent so much time on her resume. It turned out that her relentless tweaking worked. She sent her resume to top tech companies and got a response every time.

Don't ask for a resume review until you feel it's nearly final. If you're constantly asking your friends to edit a resume that you clearly haven't spent much time on, they're not going to be very happy the third time you reach out for help. If I'm doing a resume review, I want to see that you've spent time editing your resume and checking for formatting issues first.

Formatting and consistency

There are a few easy things you can do to make your resume look more professional.

- Keep capitalizations consistent.
- If you use bullet points, use only one type of bullet point; don't mix and match with arrows and circular shaped bullets.
- Your font type and size need to stay consistent throughout your resume, with few exceptions such as for a main header.
- Stay consistent in how you write about yourself—don't switch from "I" to "she" or "he" in your about me or summary section.
- Make sure the formatting is perfect.

What I've covered so far are the fundamentals, the basics you need to do at a bare minimum. The first steps. Let's look at how to avoid common mistakes.

Common Mistakes That Will Get Your Resume Rejected

I've reviewed countless resumes and I see the same mistakes again and again. This section covers what you can do to avoid the most common resume mistakes.

1. **Including personal details**

   Here are a few more things you don't need to put on your resume:

   - Age or birthday
   - Relationship/marital status
   - Gender

- A photo of yourself

- Your physical address. For security and safety reasons, don't list your full address. Put something general like San Francisco, CA, for example.

- Your hobbies. Adding your hobbies to your resume isn't necessary.

- Insignificant information that takes up space on your resume.

2. Not using a career change

If you're moving from another career to software development, don't assume you are starting from zero. List your skills and accomplishments from your past jobs and reframe them as experiences and talents you can leverage as a software engineer.

3. Not using keywords

I talked about ATS in Chapter 1 and in this chapter. ATS uses software to find candidates that match the criteria they're looking for. Too many applications don't use keywords or don't use enough of them. And it is important to be sure to include keywords throughout your resume (caveat: don't overdo it). Be crystal clear when talking about what frameworks you've used and the languages you know.

4. Readability

Don't use small fonts, pack the resume, or clutter the resume with too much information. Why frustrate a hiring manager?

5. Lazy errors

Hiring managers might overlook one spelling, grammar, or tense error if the rest of your resume is outstanding. You might even get an interview, but if you have a few errors you won't.

6. Avoid first-person pronouns

Try to say “Delivered new feature two weeks ahead of schedule” instead of “I delivered.”

7. Don’t go to the second page

Too many resumes I reviewed were more than one page. I would argue that you should keep it to one page even if you do have 10+ years of experience. The average recruiter scans a resume for seven seconds before moving on. By keeping your resume to one page, you increase your odds of the recruiter seeing the most important information.

8. Not tailored to the job

Don’t list all your experience, just relevant information about the job you are applying for. Use LinkedIn to cover all your experience and previous jobs. Before you apply, review and make sure every single section is relevant and will help you not hurt you. Be ruthless about removing irrelevant data.

9. Inflating your resume

Don’t puff yourself up. Be honest about what you’ve accomplished, what you know how to do, and your skills.

10. Sending the wrong document type

Make sure if you’re applying through a system the company uses, that you use the resume format they specify. That ensures that their ATS system will work with your resume. However if you’re sending your resume straight to a recruiter, say over email for instance, you should send a PDF version so the recruiter can’t see your tracked changes. Also save your files as YourNameResume.pdf. Recruiters, engineers, managers, and anyone else on the hiring team will have to search for your resume and it’s so much easier if your resume isn’t named Resume.pdf like everyone else that applied.

11. Listing references

Avoid saying you have references available upon request or actually listing your references on your resume—it looks outdated. Getting your references is part of the interview process.

12. Applying to too many jobs

ATS software shows recruiters how many jobs an applicant has applied to within their company. Applying to too many jobs in the same company can tell a recruiter that you don't really know which job you want, or worse—that you aren't sure of your skills.

It is not that hard to avoid these common mistakes.

If you are just starting out or are switching careers, you are probably wondering how to get a job with no engineering experience. What should you include on your resume?

My first engineering resume (with no job experience) that got me a job in two weeks

Below is my very first developer resume. I created it right after I graduated from my bootcamp.

Randall Di Ann Kanna

RK

Education

Graduated *Cum Laude* from California State University, Sacramento.

Graduated from Dev Bootcamp in San Francisco, CA.

Skills

Leading multi-member project team.

Creating a web application built solely in MEAN stack.

Implementing Agile development and Git version control in group projects.

Writing maintainable and reusable code.

Debugging and writing code to solve problems while using best practices.

Practicing MVC, RESTful, and Object-oriented design patterns.

Test-driven development and behavior driven development using RSpec and Jasmine.

Communicating a brand's content and value to drive engagement.

Growing fan bases and online platforms.

Leading social media, project management, product research and strategic communications.

Conceptualizing designs, drawing up wireframes and implementing into applications.

Proficient In

JavaScript
Ruby on Rails
HTML
CSS
jQuery
Test-driven Development
Ember.js
SQL
ActiveRecord
Angular.js

Experience

Software Developer Apprentice at Dev Bootcamp in San Francisco, CA. Completed 19-week 1,000+ hour intensive web development program focusing on skills such as Ruby, Rails, Object-Oriented JavaScript, Test-Driven Development, Agile Development and more.

Marketing associate and Facebook community manager for several companies and small businesses, including a world-renowned surgeon.

Facebook community manager and marketing associate helping Joe DiMaggio, LLC transform the legacy of the American icon Joe DiMaggio into a new lifestyle brand (Joltin' Joe) and launch of coffee line.

Projects

Team Lead at NIA. Lead a team of four on a week-long final project for Dev Bootcamp.

Developer at The Streak: Designed an app of most famous streaks in all of sports—Joe DiMaggio's 56-game hitting streak. The Streak app helps users track their progress in a series of daily action steps that build toward a larger goal.

randallkanna@gmail.com
github.com/randallkanna
randallkanna.com
linkedin.com/in/randallkanna

This resume landed me a job in two weeks when I first got started in engineering. It was far from perfect, and there is a lot I would improve upon if I wrote it today.

What I got wrong

Certifications. I didn't list any courses that I had taken, even though I had spent almost six months preparing for my bootcamp by getting certifications online. I had built really complex applications before I even attended my bootcamp, but I didn't draw attention to them.

As I noted previously, if you've just started out in your career and you don't have a robust portfolio or a ton of working experience yet, don't worry. Start with one certification. You don't even need to spend money on this. You can find a freeCodeCamp certificate, complete it online, and list it on your resume.

Didn't showcase projects. I had completed a ton of really cool projects before and during my bootcamp but I didn't showcase any of them. I briefly mentioned one, but I could have elaborated and included more projects. Remember, when you have no work experience as an engineer, you need to focus more on projects and skills.

Didn't showcase my wins. I had been chosen as a team lead at my bootcamp yet I didn't highlight that anywhere. What accomplishments do you have that should be included? They don't necessarily need to even be engineering focused.

Didn't include any open source work. There are so many great open source projects out there right now that can help junior engineers start their careers. And open source work is VERY impressive to companies. Try to find an open source project that is geared towards beginners. For instance, some repositories like Node.js will have issues that are labeled 'beginner friendly' or something similar to that. Some open source projects will have such a supportive community that they'll pair with you on a task.

You can even start with just documenting something better. Can you find a project that has an open ticket asking someone to work on the documentation? This is a small way to start contributing. It's often hard to get started when you are attempting to do something big or new or scary. Start small.

What I got right

Soft skills. If you have previous career experience, highlight the skills you've gained. As you can see in my resume, I included my previous experience working as an intern on a political campaign. I shared how I went from being an intern to helping manage interns and overseeing their work instead.

I made my skills relevant. I had no dev experience, so I included experience in Wordpress, Google Analytics, and other platforms and technical tools I had experience with.

Final Thoughts on Creating a Standout Resume

Craft a compelling story

I was promoted to a senior software engineer less than a year and a half into my career. Whenever I applied for other jobs, I highlighted how the company where I was an apprentice terminated my apprenticeship to hire me as a full-time software engineer.

I also noted that I learned new skills to help companies I worked for, including learning Objective C and Swift so I could help the iOS team, which was short-handed.

Don't be shy!

What's your biggest achievement at each job you've had? I've reviewed hundreds of resumes, and rarely did I see candidates be specific about their accomplishments. They write general statements about what they did at their job, and then wonder why they aren't getting an email back.

On my resume, I list an achievement for each position I've ever held. For instance in my first job, I detailed how I went from an apprentice engineer to a senior engineer in a year and a half. In another job, I highlighted how the project (I was one of two engineers), was featured in *Forbes*.

If you find it hard to remember your specific achievements from past positions, keep achievements up to date in your current job. Write down a quick summary of your wins each week. You can use this later when applying for your next job. Another benefit to keeping track is that your resume will be easy to write because you've been documenting achievements all year.

Showcase your passion

Most hiring managers are looking for passion. They're not just looking to fill a role with someone who has a CS degree from Harvard. In fact, some of the best engineers I've ever worked with didn't have a CS degree at all.

I once interviewed an applicant who didn't have a great start with his on-site interview. Midway through, it didn't look like he was going to get the job. Then he pulled out a personal project to demonstrate. He immediately impressed the team, and was back in the running for the position.

Fresh out of my bootcamp, I landed an interview with Apple. They said that my passion for engineering and learning was evident from my side projects, resume, and website.

A professional, well-designed resume will make yours stand out from the pile. In the next chapter, I'll explain how a solid project portfolio can go a step further to help you land job interviews.

— CHAPTER 6 —

PORTFOLIO

Having a standout portfolio will benefit you at *any* point in your career.

If you're a senior engineer and can discuss projects you've built at work, a portfolio isn't quite as important, but it still makes a difference. I've worked with some amazing senior engineers who created side projects that were generating revenue every month. Companies they interviewed with were blown away by their successful projects. Taking a project to an interview is a powerful tool. If an interviewer asks you to share something cool you've worked on, you can pull out your laptop and demo them a project.

Early on in my career, I didn't have a robust portfolio. I had an upcoming interview with a company that I wanted to work for. They asked me to create a small project. I could have completed the app in Angular.js, which was a hot framework at the time that I knew well.

Instead, to stand out, I spent the entire weekend before the interview creating a project in the less well-known Ember.js—the framework the company used. I created a very basic app that used an existing API.

Even though the application broke on the day of the interview (yikes!), the hiring managers were impressed that I had spent the extra time teaching myself how to use the *company's framework,* and I got a job offer.

What If You Don't Have a Portfolio?

Even if you don't have a robust portfolio (or any portfolio), there are ways to quickly enhance or build one.

A few years ago, I wanted a developer job in Blockchain, but I had no experience. I had taken a few courses but most of the companies that worked in Blockchain were small startups that couldn't afford to hire someone who was still learning about Ethereum and smart contracts.

So, I started building small side projects and ended up taking one of them to a company to demonstrate I was proficient. With no previous work experience in Blockchain, I was able to land a job as a developer, simply because I had created small side projects and added them to my portfolio.

Instead of feeling overwhelmed, start somewhere. Build a project this weekend or start one. Even if you don't have a lot of experience, create something for your portfolio. You can always redo it in a few weeks with all the things you learned from the first attempt.

Do you have a website? If not, build one. Your first portfolio project can be your own website. In fact, your personal site is your most important portfolio project. Your site is your calling card—the first thing that prospective employers will see. It's the way that companies will find out about your portfolio and see the cool things you've built.
Don't wait until you're ready to build a portfolio.

The best time to start a portfolio project is yesterday.

How to Plan your First Portfolio Project

Once your website is up and running it's time to build your portfolio. The first step is to have an idea that interests you. Better yet, create a list of ideas and narrow it down to one that will keep you motivated every day. Creating a Twitter clone is a cool idea, but if you're not excited about it, you won't spend enough time on the project and the quality will suffer.

Create something unique that you're passionate about. But don't get in too far over your head—your first portfolio project can be something simple. Think of a fun idea or even Google "portfolio project ideas." It will be the push you need to get started on a project that you can share with companies.

Once you've selected the idea you'll run with, grab a notebook and a pen and sketch out how you want the project to look. Remember, the easiest way to break down an app is to know exactly how you're going to approach each feature and task. Follow these steps to plan and build your project:

1. **Sketch out the flow of the app.**

   What pages do you want the user to see? What interactions? What are the main features of the application? How will it be used?

2. **Sketch out how the app will work.**

   What will you need to code? Do you need a frontend? Do you need a backend? Do you want a mobile app? Do you need a database? Will you need to deploy it?

3. **Sketch out the flow of how the different parts of your app will interact.**

   a. What tools will you need to use or learn?
   b. What frameworks will you need to complete your app?
   c. What are the languages you want to use?
   d. What databases will you need?
   e. What hosting will you use?
   f. Do you need to create a login flow?
   g. If so, what will you use to do that?

4. **Create a Trello board with a list of tasks.**

   Start with a board focused only on your MVP features. (MVP stands for a minimum viable project.) Choose only the features that are the minimum features your app needs right now so you can get it out there and start using it in your portfolio.

5. **Set concrete deadlines for the project.**

   Otherwise, you'll put things off for another day. Use the Trello board and try to estimate how long each task will take you to complete. Add that all up and set a goal to have completed the project by that date.

6. **Push your project to GitHub and consider deploying it.**

7. **Finally, create a powerful readme.**

   Your readme should include a way to run your application, run any tests you added, and explain why you created the project. You can always check out some other projects that have readmes you like and copy those.

Final Tips to Make your Portfolio Shine

Design your projects with a professional look

Appearance matters, and a well-designed portfolio will make a better impression on a hiring manager. If you have a friend who is a designer, ask for help selecting colors and fonts to improve your application's aesthetic. If you both collaborate on it, they could use it for their portfolio as well. If you don't have a friend who can design, look at sites for inspiration. You can open up the developer console and look at the structure of a web page or even the CSS.

Learn how to use Material-UI or Bootstrap or another popular UI framework and work on integrating that into your application. You shouldn't spend a lot of time styling buttons, for example, because companies generally have a solution. Prioritize building robust features and integrate a UI framework to save time. You don't need to build your own select boxes, modals, or tabbed components. Instead, focus on the important elements.

Make your app responsive. Learn how to use a tool like Flexbox and take screenshots of your app in different screen sizes. Include them in the readme of the repository as well.

Use a good color scheme and integrate nice fonts using Google fonts. You just need to find a font pairing you like (I recommend https://fonts.google.com/), and include the link in your <head> element in your app.

Feature your best work

If you have created a project that you're proud of, display it prominently on your personal site or in your resume.

It is better to have one well executed project than several projects that don't fully showcase your abilities.

Focus on your strengths

If you're stronger in frontend, try to spend as little time as possible on the backend. Use APIs that already exist and deploy your frontend using Google Firebase hosting. You don't need to create a fabulous looking frontend if your goal is to get a backend job. Similarly, if you're stronger in backend development, concentrate on that.

If you're excited about accessibility, research ways to make sure your app is accessible and discuss how you went about it in your readme.

In your readme, also talk about the deployment tools you used, and why you made the framework choices you did. Include tests that demonstrate how your core functionality is working.

Make sure to include screenshots of your application in case the interviewer is just skimming your code and not planning on running the application.

Add tests

Even if you aren't familiar with a test suite, adding tests is an important factor in standing out to a company.

If you don't feel confident in your ability to write tests, create a test file and add some comments. Write out the name of each test and the steps that the test needs to do. This will demonstrate your thought process.

Clean up your code

Don't push up messy code. Messy code is difficult to read and some people really can go crazy over indentation issues or commented out code. Make sure you stick to proper formatting and use the same spacing or tabs throughout. You can always install a linter to your code if you need it to clean it up.

Push up clean commit messages. We're all guilty of pushing up things like 'fix' or 'test,' but a company will look at your commit messages and assume you'll push up something similar to their own repositories.

If needed, use comments that walk through your thought process. Your goal, however, is to keep your code clean enough that you won't need many comments.

Now that you have a roadmap to create a standout portfolio, it's time to look over your application package and make sure you are well positioned for your first interview.

CHAPTER 7

THE JOB HUNT

After positioning yourself as a standout applicant, you are well on your way to beating the odds. Your professional, well-designed resume is likely to get immediate attention, and potential employers who explore your portfolio will get a window into your dedication and passion as an engineer.

Now, the hunt begins.

When I began my job search, my energy was lacking. I was exhausted after I graduated from my bootcamp. It was tempting to spend a few weeks after graduation napping on my couch and recovering from the most grueling event of my life. But I had spent more than $15,000 on tuition and living expenses in San Francisco, so I needed a job (and fast) to recoup my investment.

I saw that many other graduates had let their skills get rusty, resulting in a 6–12 month search before landing a job. I didn't want that to be me, so I decided to leverage the momentum, energy, and commitment I had during the bootcamp to find a job.

So, from the hours of 9 a.m. to 6 p.m., I was job hunting. I was obsessively updating my resume, finding new jobs listings, reaching out to connections, finding meetups to attend, and honing my skills.

I applied to dozens of jobs every day but the search didn't produce results until I started customizing the application to the company and sending a personalized email pitch about why I wanted to work there. (I'll expand on the pitch process in the next chapter.)

Tailoring Your Resume to the Job

I used to use an Excel spreadsheet to track the dozens of job applications I submitted. It didn't work: very few companies responded.

I realized I was making a common mistake sending out the same resume no matter what the job listing was for in engineering. I needed to tailor my resume to the specific job I was applying for.

Tailoring your resume will drastically increase the chances that it will be seen. This is critical to remember as you begin your job hunt. Remember, in chapter 1 I shared that most companies use ATS, and 75% of the resumes submitted to ATS are never read by a human being.

When you tailor your resume, add keywords from the job description, and you will increase your chances of getting past ATS.

This doesn't have to be a time-consuming process. You can create your original resume and make it quite lengthy with achievements and skills. And when you want to apply to a particular job, simply copy that resume into a new document and start editing. As you create more and more tailored resumes, you can even reuse these copies if you see a similar job to one you've applied for.

Start by running your resume and the job posting through a resume scanning site like *Jobscan*. Jobscan will give you a rating based on how your resume syncs up with a job listing by returning a list of keywords that your resume is missing.

If you have past experience with the tech stack the company uses, put that in your resume. Expand on any jobs in your background that used that framework or language and go into detail. If you haven't used anything in their tech stack, focus on how you've used similar frameworks or languages.

This can also be applied to your resume in other ways as well. If the company stresses that they love engineers who are good at mentoring, include any mentoring you've done in the past.

If they have a section on their website that states that they focus on Agile methodologies, if you have the experience, write about how your previous company used Scrum and how you improved that process. Did your previous company send you to training events to improve your Agile skills? Did you get an Agile certification? List all that on your resume.

When I apply to a smaller startup, I modify my resume to show that I've done countless technical interviews and phone screens, and created comprehensive hiring processes for companies in the past. At my current company, this was valuable because engineers at small startups need to wear several hats to fill the role.

If the company requires a cover letter (which is rare but not unheard of), be sure to tailor the cover letter to the company as well.

Getting a Job Quickly

I've had to job hunt several times in my life when I was under time constraints.

It's difficult but not impossible.

Here are a few strategies I used when I first got started:

1. I created a list of companies that I found on LinkedIn and AngelList that I was interested in, and even if they didn't have a junior developer position posted, I would reach out to them with a customized email.

2. Because I didn't have many personal connections yet, I wrote a customized message to recruiters and asked them if they had a good fit at the company.

3. I went through a six-month interview process in two weeks. I cycled through companies as fast as I could, went to meetups as much as possible, and rewrote my resume constantly when applying to different jobs.

4. I avoided companies like Google that had unrealistic expectations of bootcamp grads, prioritized algorithms expertise, and wanted CS degrees. (Companies like these are typically not good environments for new engineers anyway. Move on.)

5. I searched for companies with a focus on mentoring that had written posts on the importance of apprenticeships within tech companies.

6. If you are serious about your job search, treat the process like it's your job.

By reaching out to connections, finding meetups, honing your skills, and obsessively tailoring your resume to the job you're applying for, you can hugely increase your chances of getting noticed. Before I move on to the interview process, I'm going to go over the biggest job-hunting myths, so that you can avoid these common mistakes.

The Biggest Job-Hunting Myths

Almost all junior developers believe these myths and will be discouraged by them, but if you can turn these into opportunities you will stand out.

Myth #1: I should apply to as many jobs as possible

As I shared in chapter 5 and this chapter, when I was starting out I believed that if I applied to as many jobs as possible, I'd increase my chances of getting an interview. In case you skipped that chapter, spoiler alert, this does not work.

Myth #2: I shouldn't apply if I don't fit the posting perfectly

I lost count of the times we've hired someone who didn't fit a position 100%. Why? We liked them. Sometimes companies don't know what they're looking for in a candidate. Job postings can be flexible. If you're a junior engineer, go ahead and apply to mid-level jobs and let the company decide if you have the skills they need.

A few years ago, I was working at a small startup, and management said that they only wanted to hire very senior engineers. We posted several senior-level job positions and someone mid level applied. We ended up interviewing them because the candidate impressed us during the interview.

If you don’t match the skills listed on the job listing perfectly, apply anyway. But be realistic—if you’re applying for a tech lead position when you only have six months of experience, it is probably not a fit.

The last time I was job hunting, the role didn't completely match my experience. The company wanted more full stack experience than I had. I applied anyway and got the job. It ended up being a great fit and I taught myself the missing skills so I could do the job effectively.

Myth #3: I should take the first offer

You don’t have to take the first offer. Getting an offer is a good sign that more offers will be on the way. That said, if you like the company and the offer is good—take it. Also, if you do get an offer, let the other companies you are interviewing know you have an offer in hand. They may wish to speed up their interview process.

Avoiding these mistakes will save you time and energy in the job search process. Now it’s time to start thinking about crafting your pitch and making connections, so you can start landing interviews.

CHAPTER 8

RESOURCES FOR LANDING THE INTERVIEW

A personal connection is the fastest way to get an interview. Your chances of landing an interview are significantly increased if you leverage your connections, network, and craft the perfect pitch. This chapter shows you how.

It's About Who You Know

At least 70% of jobs are never posted.[4] This means that you might be missing MOST of the opportunities that are available when you search online for a job. And we've talked about the influence of ATS and the impact it has on your application. Simply put, there are a lot of job opportunities that you'll never know about (because they were never posted) and others that you might miss out on (because of ATS).

My first developer job wasn't posted online, and that was only an apprenticeship program. When it comes to landing a job in engineering, it's like almost any other industry: it is all about who you know.

This is why it's important to start building relationships through networking. When you have connections, you'll always be able to reach out and ask a friend for a referral.

So who do you know? Do any of your friends have connections at tech companies? Someone from your bootcamp? Teachers? If you find the courage to ask around, someone you know probably has a friend who works for a tech company. Ask for a referral.

[4]Gina Bell, "At least 70% of jobs are not even listed — here's how to up your chances of getting a great new gig," Business Insider, April 10

LinkedIn can sometimes be effective if you have a connection in common. If you don't, reach out to the company directly by sending a personalized email detailing why you like the company and why you'd be a great fit.

Reach out to an employee at the company you're interested in. Any way to get your resume into the company's hands gives you a far better chance of a response. Make the ask easy: ask if they can take a look at your LinkedIn profile or resume to see if you'd be a good fit for the company or a specific role.

Creating Your Pitch

To create a strong, compelling pitch, start by researching the company where you want to apply and finding out what makes it special.

Your email pitch should be unique to the company. Find specific commonalities between the company and yourself. For instance, I've helped create several mentorship programs at companies where I've worked and helped bring junior engineers onboard. If I learn that a company prioritizes mentorship or pairing, I include those details about my experience in my email pitch.

A general, formulaic pitch sends the message that you're not invested enough in the job to make a real effort. And first impressions make a big difference. Do everything you can to demonstrate your commitment and integrity from the start, and you will make a much stronger impression on a hiring manager.

Here's an example of an email I sent to a prospective employer:

> Hi ${companyConnectionName},
>
> I admire ${companyName} and your focus on mentorship through ${specificDetailAboutCompany}.
>
> I'm a software engineer focused on Ember.js. The reason that I'm reaching out is because I'm very interested in the ${exactPositionListed} job posting. One of the things I was impressed about is your focus on creating a UI framework. At my last company, I helped create an Ember component library that the entire frontend team used to speed up development. I'd like to discuss what I can add to your team.
>
> Best,
> Randall

You can stand out from the herd by sending emails, like the one in the previous example, that address specific aspects of the company and explain how your experience is relevant.

Interview with Madison Kanna, engineer at Keeper Security.

> **Q:** ***You got your first dev job with no experience, by sending a cold email pitch. Tell me about that.***
>
> **Madison:** "When applying to entry-level developer positions, there is so much competition. It is incredibly hard to be noticed among hundreds of applications. The hard reality is that you will be competing against hundreds of other hopeful developers, so you have to be proactive and go the extra mile. Find someone on the engineering team, LinkedIn or GitHub and reach out to them. Sent an email explaining why you would be an asset to the team. If they listed certain skills in the job descriptions, show examples of projects you've built using these skills.

I found a company that I thought had an exciting mission, and I cold-emailed them. While they were looking for someone with more experience than me, I pitched myself as an intern for them. I got the job! And was promoted to a junior developer in a few months.

It's a catch-22: employers want someone with experience, but you have to have a job first in order to get experience. Gaining experience in some way, whether through an internship, open source, or creating your projects, is a must."

Q: What are your tips for standing out as a junior developer?

Madison: "Most junior developers are looking for ways they can get a job. Instead, focus on how you can give value. Do you have an idea for a feature you would like to build for a company? Build it, and show it to them. I'm not suggesting you do free work, but provide value upfront. Help them solve problems. I struggled to find a job and wasn't sure if I would find one because I had no engineering work experience, I had not attended a bootcamp (I taught myself how to code), nor did I have a CS degree. I developed the mindset of providing value first and put it into action. By doing so, I was able to get opportunities that never would have happened if I kept "looking for a job."

The Quick and Easy Guide to Finding a Mentor

Having a mentor can help you at any time of your career. Last year, I was struggling with trying to balance leading multiple projects while learning a new framework while working on a two-person engineering team at a startup. I had a mentor who helped me get unblocked on code, navigate leading projects, and advocate for myself at the company.

Without the mentors I've had throughout my career, I would not have felt confident asking for a raise or pushing for a promotion. Having the insights, introductions, and support from a mentor who has "been there and done that" can have an astronomical impact on your career.

But finding a mentor is often challenging, especially when you first start out. It's never a good idea to reach out to someone and ask: "Will you be my mentor?" It needs to happen naturally, organically. Reach out someone you admire and ask if you can run a few quick questions by them via email. Share how much it would help you and your career. Keep the initial ask short, and then let them take the lead so you don't overstep by asking for more advice. They might not identify themselves as your mentor—they are simply being courteous and answering questions.

But if the connection evolves to a solid relationship, they can be a huge source of professional support, introducing you to companies and helping you grow your network.

Tips for finding a mentor:

- There are some sites, such as codingcoach.io, where you can find a mentor online.
- You can also start attending meetups more frequently as well, which often leads to new connections and mentors.
- If you currently work as an engineer, you can find an engineer you admire at your company and ask them if they have some time to discuss your career.

Meetups

Many companies recruit directly from meetups. But that isn't the only reason to attend meetups. I've met new friends attending meetups, found new jobs, and even landed more speaking opportunities.

Become a regular at a meetup that uses the framework or language that you would want to work on at a company. It's a small investment of time that could become a powerful tool in your job hunt.

Attend a meetup early so you can network before the session starts or when they are serving food or drinks. Once the talks start, it's much harder to connect with people.

If you get nervous approaching people at meetups, read Josh Ghent's blog post titled Networking at Tech Meetups for tips.

Conferences

Besides increasing your knowledge, you could find your next job at a conference. I've made connections with some cool companies by attending conferences regularly.

Generally, most companies will sponsor a booth or exhibit, but some companies don't want to spend the amount of money needed to be sponsors at a conference, so they'll send employees as representatives.

If your company won't pay for a ticket (or travel expenses) or you can't purchase one on your own, see if the conference offers free conference tickets for students or diversity scholarships. I've also avoided admission fees by volunteering to help out at a conference.

I wish I had gotten better at networking earlier in my career. Don't let the word "networking" stop you from meeting people at conferences. Networking is about building relationships. Build new relationships and recognize that you have something to offer even if you don't have an impressive job title yet.

Hackathons

Participating in hackathons and projects is another great way to develop a professional network. It's also a good way to feel connected. It can be nerve racking to get involved in a hackathon for the first time, but you might find your next employer at a hackathon, so jump in. You could also end up on a team with an individual who can provide a job referral.

There are also sites such as the Collab Lab where you can participate in projects and learn how to collaborate on a team remotely.

CHAPTER 9

THE INTERVIEW AND BEYOND

Finally you've made it to the interview stage. The engineering interview process can vary from company to company but generally will include a few of the following: the phone interview, the initial onsite interview, the whiteboard interview, and the take-home coding challenge.

I'm going to cover various aspects of the coding interview and continue with the whiteboard interview in chapter 10 and finally, the take-home coding challenge in chapter 11.

The Recruiter Call

The recruiter call is generally the least stressful portion of the interview process. The recruiter will tell you about the role, try to sell you on the company, and answer your questions.

But this portion of the interview isn't only about the recruiter interviewing and informing you. It is also about finding out if the company is a good fit for you. Take the time to prepare some questions about the role and company so you can get the answers you need to determine if you want to move to the next step in the interview processes. Don't try to memorize the questions—write your questions down and have them accessible in case you're nervous and have trouble remembering

During the first call with the recruiter, you should always ask what you should expect throughout the interview process.

Three tips for acing the phone screen:

1. **Turn your weaknesses into strengths**

   If you don't have a CS degree, or you don't fit the job posting exactly, find ways to turn these things into a strength. For instance, if you are a self-taught engineer or you gained experience as an apprentice, share those details and why they are strengths. Don't be apologetic for what you perceive as aspects that might be 'missing' in your resume. Be confident that you have value to add to the company.

   As I shared in chapter 5, when I went on interviews before I had any work experience as a developer, I highlighted how my previous non-engineering experience would benefit the company. I shared that I was selected as the tech lead for my bootcamp's final project, and how we had taught ourselves a new framework for the project over our Christmas break.

   I communicated during the interview that I was eager to learn and could teach myself quickly. I highlighted an experience that showed how I had done that before.

   Point to your resume's list of achievements that you want to share with the person doing the phone screen

2. **Take deep breaths and pause**

   I'm still nervous in interviews. I think most people are, no matter their level of experience. But the interviewer doesn't have to know you are. Slow down your breathing. Think of what you are going to say before you say it. Pause to articulate your thoughts, or say, "Let me think about that for a moment" to buy time while you come up with a response. Talk slower than you do in normal speech.

3. **Ask thoughtful questions**

   Be prepared with a list of questions to ask the recruiter to demonstrate that you are a serious contender for the position. This will show the recruiter that you care about the position and that you've done your research.

Passing the Technical Phone Screen

Sometimes a company has a technical phone screen before they'll invite you for an onsite interview. This can range anywhere from a few technical questions to pairing on a problem over Zoom.

Be prepared to write basic code and answer questions about what technologies you're the most familiar with. In my experience, this is one of the easiest parts of the interview to pass.

The Culture Fit Interview

Not all companies have a culture fit interview, but don't assume there won't be one. The easiest way to prepare for this interview is to write down a list of questions you might get in a culture fit interview and have a friend quiz you on them.

Here are some examples of questions a potential employer might ask to see if you are a good fit for their workplace culture:

- Tell me about a time you had an issue with a teammate or a problem on a team and how you resolved it.
- What are your career goals?
- When you come across a problem you don't know how to solve, what do you do?
- What problems are you excited about solving?

- What excites you about our company?
- Why do you want this job?
- Why are you looking to change jobs?

The Onsite Interview

Early on in my career, I was so flattered to get an interview that I made some basic mistakes. Here are a few pointers that will help you relax and be more confident from the get go.

Give yourself ample time to get there. Give yourself more than enough time necessary to get to the interview on time. If you arrive too early, you can always go to a coffee shop and review your interview prep notes.

Ask to take a break. Some companies tell you upfront that you'll have breaks between each interview. If they don't, ask for one. If you need water, ask for it.

Dress to impress. Wear comfortable, clean, professional clothing. Don't overdress or underdress. Once I interviewed someone who wore flip flops. That is too casual and may send the message that you are not making enough of an effort.

Interview with Ben Ilegbodu, principal engineer at Stitch Fix

Q: What's your preparation process for getting ready for a technical interview?

Ben: "There's always some general prep that's not related to the candidates themselves. I need to know what role they're interviewing for and what exactly the hiring manager is looking for. Once I have that information I can determine which of my collection of interview questions best fits the topic.

I also look at the candidate's resume for anything interesting and use their resume for any interesting tidbits to use as an icebreaker in the beginning. The first five minutes can be pretty tense, and I want the candidate to be as relaxed as reasonably possible. For fairness and to avoid bias, I'll ask the same question the same way so that the resume doesn't really impact the actual technical portion."

Q: If you had to pick one technical question to ask a candidate, what would it be?

Ben: "I can't share the actual question, but my focus is frontend development, typically in JavaScript, so my go-to question usually has to do with asynchronous coding in JavaScript. That usually separates those who "play" with JavaScript versus those who really develop with it."

Get ready for the "do you have any questions for us?" question

Be prepared to ask questions you might have about the company, the job, and whether the company is a good fit for you. Never say, "No, I don't have any questions" or they'll think you haven't done your homework.

Here are some of the questions you might want to ask:

1. What process does the team use?

2. Can you discuss the day-to-day schedule of an engineer?

3. How often does the company have meetings?

4. How often does the company have onsites?
   (Only ask if the job is remote.)

5. What is the onboarding process at your company?

Here are some do's and don'ts to keep in mind.

Do's.

- Come prepared with a LONG list of questions. Chances are, they're going to answer some of your questions during the interview, but you don't want to get to the end of the interview and have nothing to ask.

- Ask questions about Glassdoor reviews. If the company has negative reviews, ask how they are addressing them. This might feel uncomfortable, but try to remember that you're interviewing the company as much as they're interviewing you.

- Ask about the definition of success for this position. What do they expect in three months? Six?

Don'ts.

- Don't ask about the tech stack the company uses. You should have researched this before the first recruiter call even happens.

- You don't have an offer at this point, so don't ask about salary, benefits, or how often raises occur.

- Don't ask about the work hours. Instead, ask about the work-life balance on the team.

- Don't ask the company if they'll do a background check or look at your social media profiles.

The first few stages of the interview process can be nerve racking, but they are critical opportunities to learn details about the job you're applying for. From the company's perspective, the phone interview and the initial onsite interview will reveal a great deal about your personality, your background, and your experience. But some companies want to see you in action before the interview process is over. Up next, is the whiteboarding interview.

CHAPTER 10

CRUSHING THE WHITEBOARD INTERVIEW

The whiteboard interview, for many engineers, is the most stressful part of the interview process. I have yet to meet an engineer who feels relaxed while someone stares over their shoulder as they code. It's a nerve-racking process that can make even the most experienced engineer forget an answer to something simple.

Unfortunately, there are times when you can't avoid a whiteboarding interview. Some companies are realizing that this is an antiquated process that doesn't demonstrate a candidate's skills or knowledge, but whiteboard interviews are still being done and you'll need to be prepared.

It can be overwhelming to look at a long list of everything you need to know for a whiteboarding interview. When I first started out as an engineer, it was hard to even motivate myself to study because I felt so daunted by how much information I needed to absorb. I would take tutorials and watch videos on algorithms, but I would only scratch the surface of what I needed to do to prepare.

That's why I suggest you create a whiteboard interview study plan that will give you a measurable goal and help you to stay on track each week.

Don't look at large linked lists and trees and think you need to know it all right away. It's important to break the topics down into approachable chunks and stay consistent.

The first thing to do is to choose the language you will use.

Picking the Language

Ideally, you'll use a language that you're already familiar with. Unfortunately, that is not always an option, especially if you're interviewing at a company like Google.

Another exception might be a small startup that only wants people who are familiar in the language and framework the company uses so they can ramp up quickly.

The programming language you'll choose might also have some constraints, depending on the nature of the position. If you're looking for an iOS role, you'll probably need to use Objective-C or Swift. If you're looking for a frontend role, you'll focus on JavaScript.

For me, I've always stuck to the language that I'm most familiar and comfortable with for interviews: JavaScript.

Building Your List

The first step in preparing for a coding interview is to plan out everything by creating a comprehensive topic list. Start tackling items piece by piece, and you will create momentum as you cross off items in the list.

Your list should be specific to the type of company you want to work for and the job you want.

Preparing to interview at a large public company

A large company typically focuses on a traditional whiteboard interview. You should focus on algorithms and data structure to prepare.

Here are some potential topics you might see at a large tech company:

- Stacks
- Queues
- Sorting algorithms like merge sort
- Search algorithms
- Linked lists
- Trees

- Big o notation
- Graphs
- Hash tables
- Systems design
- Operating systems
- Heaps
- Testing
- Networking
- Recursion
- Architecture

Check out the resources chapter for my list of the best technical interview resources.

Preparing to interview at a mid-size company

A mid-size company can be more lenient in their interviews than a large public company. You'll probably be asked about real-world solutions and skills that are actually used in the job.

If you're a frontend developer, here are some examples of topics you might be expected to discuss.

- Accessibility
- Currying
- Scoping
- Prototypal inheritance
- Closures
- Callbacks
- HTML
- Await and async
- Media queries
- DOM
- Performance
- Data structures in JavaScript

- General web knowledge like HTTP requests, security, etc.
- Testing
- Apply, call, and bind
- Promises
- CSS positioning

Preparing to interview at a startup

To prepare for an interview at a startup, broaden your topics and don't worry about going too far in detail with algorithms. A startup also will be a little more flexible with a coding language. For example, I stick with JavaScript during a coding interview with a startup.

If you want to work for a small startup, jump into delivering features right away. A small startup won't be able to spend a lot of time ramping you up on the framework and language that they use. Interview questions at small startups tend to be more focused on the specific language and framework they use. For instance, if the company uses React, you might be asked how familiar you are with React Hooks and if you've used GraphQL.

You'll need to spend time figuring out your custom list. Check out the resources chapter for some of my favorite prep resources.

What about CS fundamentals?

The amount of time you spend studying CS fundamentals is dependent on the kind of company you want to work for. If you're interviewing for a large tech company, you'll need to review CS fundamentals more than you would for a small startup.

This step doesn't have to be painful. Once you build up a solid CS foundation, you'll simply have to review your notes and work on some practice problems. In the resources chapter, I've included an entire list of free online resources that will help you prepare.

Your Study Schedule

Some people can study for several hours a few times a month and go on to crush the whiteboard interview. I'm not one of those people, I need to prepare by practicing every day. Interviewing prep is a stressful process, but you can make it much more approachable if you set a schedule that works for you and practice every day.

1. Divide the topics into the amount of time you have to prepare.

You've made a list of the topics you need to study in the previous section. Now, break it down and select the topics that you'll focus on every week. How much time do you have to prepare for your interview? For instance, if you have six weeks, you can divide your topics into segments with milestone goals at the end of each week.

2. Create a daily schedule.

Determine how many hours you have each day to study. Don't believe you have an extra 1 to 2 hours a day? Create it. Stop watching television, browsing Instagram, or wasting time online. Give it all up until you get a developer job. Doing so will be extra motivation to get the job so you can resume your normal life. On weekends, find a way to schedule more than 1 or 2 hours to make even more headway.

After you've created extra time to focus on interview prep, create calendar blocks so you can't book anything else during those time slots. This is your time to focus and prepare for getting your new job. Even if it is only an hour each day, that might mean you have time to work on a few Leetcode questions in that hour. That's 30 hours in one month.

Here's an example of a weekend study schedule:

Saturday

9 a.m. to 11:00 a.m. - Select one programming topic from the list I created above and spend two hours studying it. Watch videos and read books and blog posts on that one topic. Create flashcards and take notes. Commit to pomodoro sessions with frequent breaks.

11:00 a.m. to 12:00 p.m. - Solve 1 or 2 easy algorithm problems and one difficult one after. I love starting my algorithm prep with an easy win. I get a burst of energy when I can solve a few easy problems, which gives me the momentum I need to tackle the challenging ones.

Before you begin your study session, make sure you have your resources ready to go. Don't sit down to study and then spend 20 minutes pulling things together to get organized. Be ready to go and will make much better use of your time with a stronger focus.

Interview with Kyle Shevlin, senior software engineer at Webflow

Q: How do you prepare for an algorithm interview? Do you set a schedule?

Kyle: "I tell people I'm allergic to calendars, so I'm probably not the person to ask about scheduling, but I think a wiser person than me would set a schedule. If you're doing a period of interviews, having your brain in the right headspace can be a benefit, and even devoting 20–30 minutes to a problem each day could be beneficial.

That being said, I think you prepare for algorithm interviews up front in your career and require fewer refreshers and updates to your knowledge over time. Putting in the work to learn some data structures and basic algorithms will train your brain to break a problem down into its algorithmic components. Once you've learned to do that, it will become so natural that you won't need to retrain yourself to do it again.

Being good at this material really boils down to understanding at a fundamental level how some of this stuff works. When you know the fundamentals you can apply that knowledge to specific instances. You start to understand why memorization might be useful here, but not there. Why an array is fine in this situation but a map would be much better here. Learning these basics takes you from writing code that works to writing code that works well.

I want to add one more thing. Unless I'm going for a position at a FAANG company, I don't find there’s much technical prep required beyond understanding the goals of the company and how technology serves those goals. I've put in the work up front to be technologically proficient. Your readers are doing the same by reading this book. I also consistently learn in public and create content that demonstrates my skills and value to others.“

How to Approach a Whiteboarding Problem

If you follow my system for solving a whiteboarding problem and get enough practice, you won't feel out of your depth during an interview. Before I had a system, I felt chaotic and rushed when I had to complete a whiteboarding algorithm. Being a pro at whiteboarding and solving problems is just like driving a car. You need to get enough practice and have muscle memory.

Do you understand the problem?

If you understand a problem, it should be easy to solve. Make sure you understand what the interviewer is asking. Ask clarifying questions.

This can help a number of ways. It can help you identify edge cases, and it can help you make sure you understand what they are asking. It's also your chance to ask questions and demonstrate your analytical thinking and skills.

For example, what if you're asked to write a function that checks if a string is a palindrome?

Write out the skeleton for the function. This will help you stay on track for what needs to happen later.

```
const checkPalindrome = () => {}

checkPalindrome("racecar") // should be true
checkPalindrome("Hello") // should be false
```

Writing out the function like this also helps you inquire about the test cases. For instance, should "racecar" be true? Should "Hello" return true? This isn't the most performant way to check if a string is a palindrome by the way.

Use this time to chat with the interviewer about a few different ways you might approach the problem.

Pseudocode first, whiteboard later

The most important thing you can do when whiteboarding a solution to an interviewer's question is to start with pseudocode. Write out the steps that will solve the problem one-by-one in English. Don't code anything yet.

Let's use our palindrome example again. Here's what the pseudocode will look like:

```
// Create a function that checks if a string is a palindrome
// Split the string
// Reverse the string
// Join it
// Check if it equals the original string
```

Next ... start writing the code.

DISCUSS HOW YOU COULD IMPROVE YOUR CODE

Ideally, you would write a performant version of your code the first time. But, this isn't always the case. After you've completed your solution, take a second glance at it. See if there's anything you can refactor or improve.

Kyle Shevlin

Q: What's the one thing a reader should know to do well in a whiteboarding interview?

Kyle: "I believe that a technical interview is designed to test your communication skills as much, if not more, than your problem-solving skills. In my experience as an interviewer, candidates do not keep this in mind. Software engineers are often so focused on fixing problems that they fail miserably at communicating with the interviewer and gathering all the information necessary to work on the problem at hand. Given how much communication is involved in your daily work, this is important to get right in the interview.

Your job in an interview is not to solve a technical problem. It is to demonstrate what you'd be like to work with as a coworker, a teammate, a manager's report, etc. This requires excellent communication.

In a technical interview, part of that communication is being able to express your thought process and decision making as you solve the problem. The interviewer cannot be in your head. You need to grant them access to it so that they can assess your skills. Learning to communicate this way comes far easier to some than others. If it’s not easy or natural for you to talk about your thought process as you solve a problem, you need to practice and you need to figure out how to be effective.

Find a rhythm that works for you. Communicate what your intentions are, write out that part of the solution. Communicate again, write again. What's important is that by the end of the interview, the interviewer is confident that you can think through problems and that you'd be pleasant to work with. If you get the problem done in the time allotted, that's icing on the cake.”

My Four Hacks to Whiteboarding Interview Prep

At the first company I worked for, my boss did me a huge favor. He made me whiteboard in front of several people and the other junior engineer at the time. Sometimes he made us do it at the same time side by side.

Back then, I thought it was cruel and unusual punishment. Over time, I gained confidence and started to look forward to whiteboarding.

Lesson: If you can prepare for the whiteboarding interview enough, you will ace the interview.

1. Buy a whiteboard

The first time I had had a whiteboarding interview scheduled, I prepared by writing code on my computer. I thought that would be enough. It wasn't and I didn't make that mistake again.

Writing code on an editor and writing code on a whiteboard are completely different. My text editor auto finishes some lines of code for me, which I had grown accustomed to. As a result, I didn't feel confident writing a function on a whiteboard.

A whiteboard will help simulate the actual interview.

2. Recreate the interview experience

A friend can help simulate a whiteboarding interview. Give your friend a list of questions that you might need to answer during an interview and ask them to pick one at random.

Use your whiteboard and have the friend stand near you and watch you solve the problem. Set a time limit and pretend that an actual engineer is watching. Whiteboarding answers to coding questions over and over again helps you gain confidence for the real thing (bonus points if your friend is an engineer).

3. Talk out loud

It's not easy to think, whiteboard, and speak at the same time. When first starting out, it won't feel natural to you. Unfortunately, for a coding interview, that is exactly what the company is asking you to do: think, whiteboard, and present/speak.

The interviewer wants to hear your thought process and how you solve problems. When you're practicing on your whiteboard, start detailing each step you're going to do when you're alone to practice talking out loud as you write.

4. Find an accountability buddy

If you have a friend who is also preparing for coding interviews, make them your accountability buddy. Share your plan for studying for the week and ask your friend to help you stay on target. Do the same for them.

For me, one of the benefits of attending a coding bootcamp was that I made new friends who were also job hunting and preparing for interviews at the same time, which made it easy to support each other.

If you don't know anyone who is job hunting, find an accountability buddy online who is on the same path as you so you can help each other stay accountable.

(I've included a list of resources for technical coding challenges in the resources chapter.)

Interview with Melissa Roman, senior software engineer at Eventbrite

Q: How do you prepare for the dreaded technical interview?

Melissa: "Practice. I know that answer might not be what you want to hear but it's really the only thing that helps prepare you for a technical interview. Regardless of what type of engineering job you're applying for (backend, frontend, SRE, etc.), there are tons of resources online to help you practice.

Keep in mind that you can't predict the questions you'll be asked, but you can practice a variety of questions and get used to thinking aloud and trying to logically solve algorithm and coding problems. Research and ask the recruiter what the interview process is like—typically companies want you to succeed, so the recruiter is often willing to give you as much information about the format as he/she is allowed. And since most (maybe all) processes include some form of onsite technical interview where you have to whiteboard or actually code through problems, practice doing just that.

On a whiteboard, there is no autocomplete or error-highlighting, so get used to writing out functions and algorithms without those amenities. If you have a whiteboard to practice on, even better!

As you practice, talk through your thoughts and logic (even if it's to yourself)—that's often what your interviewers want to hear and the main way they'll be evaluating you. Try to first solve for completeness, and then go back (or even just verbally note as you go) to places where you can refactor to make your solution more efficient."

Q: Any tips for staying calm during a whiteboarding question?

Melissa: "The more you practice, the more natural it will feel to whiteboard during a technical interview. First, make sure you understand the question and what your interviewers are looking for—ask clarifying questions.

Before diving into the solution, write down some pseudocode to help organize your thoughts and also to refer back to as you solve the problem. Take a deep breath. Leave extra space in your solution—you don't want to get hung up on a missing curly brace, weird indentation, or just running out of space.

Try to keep it neat, both so you don't confuse yourself and also to make it easier for your interviewers to follow. Sometimes it's even helpful to run through a test case before you finish to see if your thought process is returning what you expect. And this also shows that you think about testing."

Q: What can I do when I feel overwhelmed about preparing for an interview?

Melissa: "Start small. It's easy (and natural) to feel overwhelmed—there are so many resources, questions, answers, etc. So, start small. On day one of prep, start with just one small problem (maybe even one you've done before), one slightly harder or newer problem the next day, and so on.

Once you get started and it becomes a habit, you won't feel so overwhelmed. There will always be more that you can practice, but once you get started, you can eventually narrow in on some more questions that are more specific to the role/framework/area that you'll be applying for.

Typically most companies want to see knowledge and problem solving in the fundamental areas. (For example, most frontend roles will have interview questions about fundamental JavaScript.) Start with those questions and once you get the fundamentals down to a comfortable level, add in framework-specific questions (i.e. React and Redux)."

My Secret for Getting an Offer in Two Weeks

Earlier in my career, I was working at a company where I wasn't happy anymore. My manager and I were constantly clashing. I wanted to leave as soon as I landed another job.

At the time, algorithms weren't my strong suit so I knew it would take me months to be ready to complete an interview at a company that prioritized complex algorithm-focused interviews.

The solution? I focused my job hunt on companies that prioritized real-world interview problems instead of algorithms. I knew my actual coding skills could land me a job.

I began by asking recruiters about the engineering interview process before even scheduling a call. And I searched through Glassdoor and Twitter to see which companies focused on interview problems that I knew I could solve. I also asked friends if they knew of companies that fit my criteria.

I created a list of companies that fit the bill and started interviewing. I had multiple offers in a few weeks. If you have the time to prioritize algorithms you won't need to use this strategy. But, like me, if you feel you have to get a new job quickly, try this approach.

If you're in a rush to get a job, decline to move forward with companies that focus on five-hour onsites and whiteboarding challenges (if you aren't strong at those types of interviews). Don't waste your time on whiteboard interviews you aren't ready for. Focus on companies that are a better fit for your skill level now.

So you've crushed the whiteboard interview, and you are well along in the process, but there's more work to do: the take-home coding challenge.

CHAPTER 11

CRACKING THE TAKE-HOME CODING CHALLENGE

I'm a big fan of take-home coding challenges in the interview process because it gives candidates a chance to shine in a no-pressure environment. It also helps recreate how you would actually work on the job minus the time constraints.

Knowing how to navigate the take-home challenge can get you the offer.

What Do I Do If I Can't Finish the Take-Home?

Companies can underestimate how long it will take to complete a take-home challenge. In fact, a few years ago, I was interviewing with a company that gave me a take-home challenge that was nearly a complete app.

Here's a little preview of the take-home challenge:

- Create two applications using React *and* React Native.
- Deploy the web AND mobile applications.
- Create multiple pages and buttons to navigate through the application
- Implement a 3rd party SDK
- Pull from a restful API of your choice
- Use React components from the community

I could not believe it. I later discovered that the company had no engineers on staff yet and had just pieced together what they thought a take-home should look like from Google searches.

I chose not to complete the take-home.

Be smart when you decide which take-home challenges you will complete. A take-home challenge should take you a couple hours to a half day at most. If a company asks you to devote more time than that, they might simply be trolling for free work.

If you're facing the deadline the company has given you for the take-home and you still haven't completed it, email them and send your progress so far. Include a list with the steps you would take to finish the application.

After you've done that, spend time finishing the challenge and email it to them after the deadline. This is a very successful tactic to buy yourself a little more time, honestly and with class, while demonstrating how seriously you take deadlines.

Four Things Companies Look for in a Take-Home Challenge

Companies are vetting up to dozens of candidates every week, so it's important that you know what companies are looking for in a take-home.

1. How you approach the problem

Companies can save anywhere from thousands to millions of dollars if an engineer can break down a problem and tackle it in a more efficient way.

That's why you need to demonstrate your thought process to the company. Include a readme with the take-home challenge that discusses how you approached the challenge and broke down the pieces.

Share your ideas and what you would do in the app if you had more time.

2. Did you write the code?

Sometimes the company will ask you to have a second onsite interview after the take-home. This is your time to shine. Show them that you wrote the app (and I hope you did!) by walking through each feature.

Companies could also ask you to pair onsite and add a feature to the application. The company is looking for a number of things when they ask this.

They want to confirm you built the application. And they want to know if you can confidently add a feature to it. This adds some stress to the interview for you, but you can use this to your advantage--you wrote the take-home application so you know how to add a feature to it quickly.

Adding an onsite element to the take-home also is the perfect time to show off your debugging skills. The company might want to see if you can quickly debug a problem and how sophisticated your thought process is when you're trying to figure out a bug.

Tip: If you're particularly proud of a feature you wrote in the application, don't be coy about bringing it up to the interviewer.

3. Best practices

Companies will also be looking to see if you follow best practices. Did you write an app that was full of bugs because you had a roundabout complex solution? Or did you write something that was simple yet solid?

4. Questions

Ask clarifying questions when you receive the take-home challenge.

A company will want to know if you are engaged. Companies, especially startups, want employees who will point out poor tech decisions and help the company prioritize work.

On the other hand, once you've asked a few questions, try not to email them again until you've completed the challenge. This will tell the company that you're confident and can take work and run with it. If you run into issues, try to solve them on your own. If you can't, feel free to shoot out an email, but don't be in constant contact with the company about a take-home challenge.

Here Are the Best Ways to Debug Quickly

Know how to reproduce the bug. It's much easier to fix a bug if you know how to recreate it.

Don't be afraid to Google. Learn how to Google for solutions, efficiently. Try to use keywords as much as possible to keep your search broad. Similarly, don't include your file path, as this will limit your search results. If the error is coming from a specific library, add the library name to your search.

Memorize HTTP error codes. If you know at the top of your head what the error code is, this will go a long way when debugging an API call. For instance, if you see a 401 code, you'll immediately know that your request cannot be authenticated. HTTP cats is a fun cat-themed site that you can use to memorize error codes.

Use debugger statements or log liberally. Many senior engineers I know don't know how to use the debugger effectively. Learning this skill will help you solve problems like a pro.

Many times I've been stuck on a bug only to discover that a section of code wasn't even being run. It's a running joke at work that I console.log ("Rya") (my dog's name) in the app and sometimes forget to remove it and commit the code.

Learn how to use the developer console in your preferred browser well. I love using Chrome but this is a personal choice. Whatever browser you do pick, take a course and read some tutorials for your preferred browser.

Become an expert-level debugger. Learn how to step through your code and add breakpoints. Step in and out of functions. And check the network tab.

A Standout Take-Home Challenge

You shouldn't spend too much time making the take-home the best thing they've ever seen, but you should spend time making it special in some way.

Once I was interviewing for a senior role and the company asked me to complete a take-home to assess my coding ability. They sent me an assignment with an optional frontend challenge and asked me to complete it if I had the time.

I wrote the backend of the assignment but I didn't have time to complete the optional portion. I sent in the take-home and explained to the company that something came up and I was unable to complete the frontend of the take-home challenge in time.

Later in the week, I completed the frontend portion and emailed it to the company.

The company was blown away that I had followed up, and I got a verbal offer the next day.

Going above and beyond on the take-home challenge

Find your niche for the take-home and make yourself stand out from other applicants. If you're just starting out, use what excites you and include that in your take-home. Here are things to keep in mind:

- If you have the time, add an extra feature that showcases your skills. For instance, I love tests, so I always include a test section in my take-home challenges. Every feature I write in the app has robust tests.

- If you specialize in accessibility, include a section on how you made your app more accessible. For example, why you used a button instead of a link.

- If you're an expert in performance, write up a section on how you made the app performant.

- If you love CSS, don't use Bootstrap or Foundation and instead, write your own custom styles.

The great part is you can reuse this extra work later in other take-home challenges.

Write tests. As I mentioned previously, I include test sections in my take-home challenges. If you have the time, try some TDD. Even if it's not spelled out as a priority for the company, it's generally a good idea to include some tests.

When I interview an engineer I always check to see if they include tests. Even a few tests gives the candidate bonus points in my book.

Include a thank you. Remember that the engineer reviewing your coding challenge will spend valuable time reading your code. Send a thank you email, letting them know you appreciate their time.

Include a README. This will be the easiest part of the application to complete and could win you more bonus points. Adding a README will help the engineer running your code immensely. They'll be able to see how to run the application and tests quickly.

Also include a checklist section of all the tasks you completed. This shows at a glance that you understood the requirements and took care of them. Include paths to a testing file as well.

If you added a bonus feature, make sure you highlight that in your README so it isn't missed.

Make it accessible. If your coding challenge is focused on the frontend, show that you know enough about accessibility by making it inclusive. This will also provide an opportunity to showcase your skills when you go over the code with the interviewer.

You can use the wave browser extension to quickly check if your work is accessible. And this link provides another list of tools you can use to check your work. You can also discuss accessibility testing tools in the interview.

Before you submit your challenge:

- Check for color contrast. This is a huge issue for people who are color blind and people with poor vision.
- Check that there is no flashing (for people who have seizures).
- Add image alt text.

Make your code performant. If you know a lot about writing performant code, show that in your take-home challenge. And in the interview later you can discuss how the code you wrote is performant and why.

Eight Things That Will Get Your Take-Home Rejected

1. Submitting messy code

Many developers have pet peeves about how code looks. Generally when you start at a new company, you should follow their style when it comes to tabs vs. spaces, indentations, and code comments. Some of this is nitpicky, but it's important that your coding challenge reads well and looks good.

2. Indentation issues

Make sure that you stick to the same indentation through the app. This will make your code easier to read and will showcase your attention to detail.

Pick tabs or spaces and stick to it throughout the application.

3. Forgotten debugger statements or console.log statements

I shared previously that I do this all the time. It's a running joke at my current company when people find my dog's name throughout the app because I use it as a marker in the code.

But in a coding challenge, make an extra effort to remove all of these statements to keep your code clean.

4. Messy leftover code

I've done this, but it's sloppy and should be avoided. It also wastes the time of the engineer who will be reading your take-home challenge.

Remove your old unused code. I made this mistake when I was a junior engineer, thinking that it might show my thought process. Instead, it just made the file harder to read for the engineer examining it.

5. Not allocating your time well

Submitting a complex application that isn't finished < Submitting a simple application that is completed.

If you manage your time well and make the application a little more impressive, that's good, but it's best if you make finishing the application a priority.

If you email the company twice that you need more time because you created an overly complex application, it will count against you. Use the requirements and consider them an MVP.

Make it work *first* and then make it look good.

6. Unreadable code

Keep your code as clear and simple as possible. The best written code is easy to understand and to read. Use clear variable names and function names. Keep it modular and clean.

7. Not being honest about your skills

If you don't feel confident in completing the take-home challenge, be upfront about it. You might save yourself hours working on a take-home challenge for a company that is looking for someone more senior.

I was interviewing at a company for a full stack position with an emphasis on frontend, and they asked me if I felt confident completing the backend portion of the challenge in the time allotted.

I told them I didn't feel confident enough to do the challenge, but I was willing to try. Instead, they were pleased I spoke up and paired down the challenge. By speaking up, I saved myself hours of time that I could have spent trying to figure out extra work that they didn't really care about.

8. Sending in code you don't understand

If you're a junior engineer, don't submit code you don't understand. If you copied and pasted a few lines from Google, make sure you understand it.

One company I worked for had a candidate apply that was junior for a non-junior role, but they decided to take a chance and sent the applicant a take-home.

We sent the candidate the take-home challenge and when they sent it back, it was copied from a popular tutorial. We knew this because they had accidentally left a comment from the tutorial labeling where they found it online.

We clicked the link and saw the tutorial. One half of the code was copied and pasted from the tutorial.

They didn't get the job.

Chances are a company will do their research, so don't cheat. Do the work yourself.

CHAPTER 12

THE OFFER

Phew, you did it! You worked hard through the lengthy interview process and now have received a formal offer. Congrats!

How do you know if it's the best offer? My first offer as an engineer wasn't much regarding compensation. I didn't have a CS degree or years of work experience as a developer, nor was I experienced in job negotiations. And at the time, I didn't realize that fellow bootcamp graduates were getting six-figure offers.

I also didn't negotiate a year later when I discovered that other engineers at the company were getting paid more than I was. At one point, a kind manager tried to help me get retroactive pay, but it didn't happen. I estimate that I lost at least $50,000 in pay over a six-month period.

My advice is this: If you aren't thrilled with the offer, don't accept it. Assess it.

Let's look closer and how to do that.

Compensation

Salary

Once you start working for a company, they aren't incentivized to pay you more in raises or bonuses. The largest salary leaps you are going to make in your career are when you switch jobs.

Once you sign for a fixed salary, you're committed to that amount for the foreseeable future. Be sure when you sign that you can comfortably live on that salary.

Equity/stock options/stock grants

If you go to work at a public company, you'll normally receive stock options that you can sell after you vest. Typically, you'll be vested after four years and you'll vest a certain percentage each year.

A startup will sometimes offer you equity. There is a lot to like about being part of a startup, but the equity you get isn't worth anything unless the company goes public or is acquired. So take equity from a startup with a grain of salt. Nothing is a sure thing.

Bonuses

A bonus is dependent on company performance, your performance, seniority, or perhaps a company policy such as a holiday bonus for all employees.

Signing bonus

Sometimes you'll get an offer with a monetary bonus for signing. If you have a signing bonus offer from one company, you could ask for a signing bonus from another company in order to incentivize the second company and gauge its interest. But a bonus offer usually comes with conditions. For instance, if you leave the company prior to the one-year mark, you may have to return a portion of the signing bonus.

Perks

One great benefit of working in tech is that some companies will offer benefits that are almost equal to your fixed compensation. It's not money in your pocket, but depending on the benefit, it can save you a considerable amount of money over time.

Meals: Companies will sometimes offer catered meals. When I was working at Eventbrite, they offered several meals a day. This benefit reduced my grocery budget considerably.

Tech stipend: If you work remotely, some companies offer a monthly stipend that you can spend on your home office or pay for a coworking space. If you work at an office, they'll generally give you a new computer. Some remote companies will give you a new computer as well as the benefits mentioned above.

Wellness stipend: A wellness stipend can be used for massages or gym memberships or other wellness services or products. This varies from company to company.

Transportation: If you work at a large company and will need to commute, the company may provide a transportation fund. This could include anything from a discount on the train to your office to Lyft credits, or straight up additional money to your salary for the commute.

Further considerations

Will you be happy with the base salary? You will no longer have the same negotiating power after you sign the offer. Generally, a company won't increase your base salary by much unless you get a decent promotion. The most significant raises to your base salary occur when you switch companies.

Will the company offer you growth? Do you have opportunities to get promoted? Depending on what you want from the job, this could mean more than the salary.

When I accepted my first engineering job offer, the title and salary were lower than other companies were offering me at the time. The title was apprentice software engineer and the salary was negligible if you calculated the amount I would need to pay in rent in San Francisco.

The flip side was that the company offered me a great deal of mentorship and growth potential. They promised pairing sessions to focus on learning without the expectations of producing code for the company.

In the long run, I became a senior software engineer sooner and started making far more money than I would have if I had taken another offer with less chance for growth, training and mentorship. That was more important than a larger salary.

Does this fit in with your career plan? If you want to have an exceptional career, make strategic career moves each step of the way. Where do you want to be in two years? Then ask yourself if accepting the new job will help you get there.

Will it be a culture fit? Research the company on Glassdoor and ask past employees on LinkedIn what they thought of the culture. If it turns out you don't feel comfortable working at the company, taking the job will affect your job performance and mental health.

How stable is the company financially? Make sure the company can deliver on its financial promises.

How are the benefits? Will the benefits fit with your health and financial situation? Do they have dental and vision insurance? Does the company offer a 401K? Are adequate vacation and sick days provided?

Negotiating

The first offer is never the best offer. When a company makes an initial offer to a candidate, they often have more money available for a counter offer in case the candidate intends to negotiate or has an offer from another company.

Try to determine how much more is fair. You can use sites like PayScale to find a rough estimate of your job title and location. You can determine what the company is paying other engineers at your level on Glassdoor.

Never reveal your current salary. No matter how much a company pushes you to share your current salary—don't do it. It's illegal in some states for them to even ask. How do you keep the conversation going without sharing your salary? Say, "I'm sure we can find something that will satisfy both parties." And repeat it as often as you need too.

State the right reasons why you deserve more. Don't bring up personal reasons why you need a higher offer. Your student loans or bills should never be mentioned during a negotiation. Instead, list specific examples on what you'll bring to the company and exactly why your talent and skills warrant a higher offer.

Will the company rescind its offer if I ask for more money? No! You won't be punished for advocating for yourself and what you're worth. And on the miniscule chance that the company does rescind the offer because you asked for a little more money, you will have saved yourself from working for an undesirable company.

What if the company says no when I negotiate? If the company isn't open to negotiating a compensation increase, you could negotiate for other benefits. You could ask for additional paid time off, better stock options, or a stipend for personal learning/conferences.

Interview with Shawn Wang, author of *The Coding Career Handbook*

Q: In your book, The Coding Career Handbook, you talk about how you were recently able to negotiate an additional 50k in salary. How do you avoid stating what you're currently making when a recruiter asks for your salary expectations? Any tips?

Shawn: "Yes, I say it's not legal to ask in my state. It's not legal in 18 states! Fortunately, in my case they didn't actually ask because they're professionals, but they all want to know. I highly recommend reading Haseeb Qureshi's Ten Rules for Negotiating. He suggests that you treat all your asks—especially salary expectations—as closely held pieces of information.

Even more important, make sure that the role you're being considered for matches your expertise, as no amount of negotiating can undo being underleveled. Also, have a good BATNA (Best Alternative to a Negotiated Agreement). This could be having another job offer in the works, remaining in your existing job, or pursuing something else you want to do with your life.

I write for early career devs, and I caution them not to expect to negotiate too much. When approaching and anticipating your salary needs, optimize for the company you can grow with long term, rather than trying to get the highest number short term.

When you're just starting out, know your market value and make sure that you get it. You can educate yourself on what this is by using Levels.fyi, TeamBlind, and the high end of Glassdoor. If you're working remotely you can use open salary calculators from Buffer, Krit and GitLab to figure out what you'd get there. If you're in a niche like infosec or data science, certain subreddits will run salary surveys you can check, and if you work at an agency, check out the rates by country. Over time you'll gather enough unique experiences, skills, and network that you can credibly seek an above-average market value."

You're Not Married until There's a Ring

My coding bootcamp gave me a piece of advice that I've never forgotten. They said that until you're married, you keep dancing with other partners. Translation, until you have a signed offer in your hand, don't assume anything is finalized. Yes, the saying is old-fashioned and perhaps a little dated, but it still *rings* true.

Anytime I've forgotten to follow this advice, I've gotten burned. A while back, I was working at a company that suddenly lost funding. I was out of a job unexpectedly and it was at the worst possible time to look for a new position.

When I did interview with a company that I was excited about, they asked for a lot of work for what should have been a simple take-home challenge. They kept stringing me along as I continued to jump through hoops. They gave me hope that I was going to be hired shortly, and I assumed the job was a *sure thing*.

This went on for a few weeks. During this time I told other companies I was hired so I wasn't available. All this while I was unemployed.

I told the company that I needed an answer by Friday EOD.

Guess what? They didn't hire me. I don't think they ever intended to. I had wasted so much time. And I had to start my job hunt over.

I had forgotten the adage my bootcamp shared with me, and I never made that mistake again.

Until you have a signed offer from a company in your hands, don't assume you do. No matter how well you think things are going, keep interviewing and do as many take-homes/onsites as possible.

Not only will this save you from thinking you have a bona-fide offer when you don't, this will also help you get multiple offers at the same time and use them to negotiate later.

CHAPTER 13

CONCLUSION

Becoming a standout developer is within your reach. But achieving professional goals and advancing on a career path only happens when you make the commitment and put in the effort.

Remember that growth isn't necessarily a comfortable process. I may never enjoy speaking in front of a large group of people or advocating for a promotion, but that doesn't stop me. My sincere hope is that this book has inspired you to do the same. Stretch out of your comfort zone and you will be shocked by the doors that will open.

Whether you are a recent graduate with a CS degree, a self-taught coder, or a senior engineer, you can't afford to hold yourself back from realizing your true potential.

Follow this roadmap to help you beat the odds so more companies will discover you. Congratulations! You are well on your way to positioning yourself as a standout developer.

CHAPTER 14

RESOURCES

The Best (Free) Resume Resources

Resume templates

Canva has great resume templates available for free.

MyPerfectResume will take the stress out of building a resume. They'll help you pick a resume template based on your experience and help you fill out each section. You can even upload an existing resume to pre-fill data.

Resume.io is another easy way to quickly build out your resume. They have some great looking templates and it's very easy to customize and expand on certain sections if you don't have experience as a developer yet.

Creddle - A cool way to customize resume templates and even automatically use less space or more depending on how much information you have.

ResumeWorded - A cool resource for some ATS-friendly templates.

Resources for editing your resumes

Grammarly is a free tool that you can use to edit your resume and check for any mistakes. This is useful for editing blog posts and books as well.

Free resources for improving your resume

ResumeWorded is a site where you can upload your resume and receive a score along with information on items you could improve.

ZipJob offers a free resume review from an actual person.

Free ATS resources

JobScan will help you optimize your resume's ATS score. You can run your resume through JobScan and it'll give you a match score to see how likely it is that you'll make it through the system.

Skillsyncer helps identify what keywords are missing from your resume.

Top Resume will check if your resume will get through ATS.

Comprehensive List of Job Boards

Popular Job Sites

GitHub - https://jobs.github.com/
Mashable - https://jobs.mashable.com/jobs/
Indeed - https://www.indeed.com/
StackOverflow - https://stackoverflow.com/jobs
LinkedIn - http://LinkedIn.com
Glassdoor - https://www.glassdoor.com/index.htm
Dice - http://dice.com/
Monster - http://monster.com
Simply Hired - https://www.simplyhired.com/
Hired - https://www.hired.com
Toptal - https://www.toptal.com

Remote Job Sites

FlexJobs - http://www.flexjobs.com/
We Work Remotely - https://weworkremotely.com/
Remoteok - https://remoteok.io/remote-dev-jobs
Stackoverflow Remote - https://stackoverflow.com/jobs/remote-developer-jobs
Working Nomads - https://www.workingnomads.co/remote-development-jobs

Remote.co - https://remote.co/remote-jobs/developer/
Remoters - https://remoters.net/jobs/software-development/
JS Remotely - https://jsremotely.com/
Frontend remote jobs - https://frontendremotejobs.com/
IWantRemote - https://iwantremote.com/
JustRemote - https://justremote.co/remote-developer-jobs
Outsourcely - https://www.outsourcely.com/remote-web-development-jobs
Pangian - https://pangian.com/job-travel-remote/
RemoteLeads - https://remoteleads.io/
Remote Talent - http://remotetalent.co/jobs/
Remotive - https://remotive.io/remote-jobs/software-dev
Daily Remote - https://dailyremote.com/

Startup Job sites
AngelList - https://angel.co/jobs
Product Hunt - https://www.producthunt.com/jobs
Startup Hire - https://www.startuphire.com/
Startupers - https://www.startupers.com/
YCombinator - https://news.ycombinator.com/jobs

Misc Job Sites
The Muse - https://www.themuse.com/jobs
Tuts+ - https://jobs.tutsplus.com/
Krop - https://www.krop.com/
PowerToFly - https://powertofly.com/jobs/
Developers for Hire - https://www.developersforhire.com/
Codepen job boards - https://codepen.io/jobs/
Joblist.app - https://joblist.app/
Fullstack Job - https://fullstackjob.com/
Authentic jobs - https://authenticjobs.com/
Jobspresso - https://jobspresso.co/
Jobs in Europe - https://landing.jobs/
TripleByte - https://triplebyte.com/
Vettery - http://vettery.com/

Language/framework/field specific
iOS Dev Jobs - https://iosdevjobs.com/
React job boards - https://www.reactjobboard.com/
Javascript JS remotely - https://jsremotely.com/
Vue jobs - https://vuejobs.com/
Python Jobs - https://www.python.org/jobs/
Blockchain job board - https://www.blockchainjobboard.org/
JavaScript job XYZ - https://javascriptjob.xyz/
Ember - https://jobs.emberjs.com/
RubyNow - https://jobs.rubynow.com/

Junior dev jobs
JrDevJobs - https://www.jrdevjobs.com/
Stackoverflow Junior jobs - https://stackoverflow.com/jobs/junior-developer-jobs
Protoge - https://protege.dev/

Women-focused job boards
Women Who Code - https://www.womenwhocode.com/jobs

Freelance job postings
Freelancer - https://www.freelancer.com/jobs/software-development/
Upwork - https://www.upwork.com/
FlexJobs - https://www.flexjobs.com/jobs/web-software-development-programming
FreelancerMap - https://www.freelancermap.com/
Gun.io - https://www.gun.io/
Guru - https://www.guru.com/d/jobs/

Other unique places to find a job

Slack communities - Some of the best job opportunities I've found were on slack channels on frameworks or languages. Almost every tech-focused open slack community has a jobs channel where members post jobs.

Discord - Some Discord communities also have job boards.

Twitter - Twitter is full of people posting job opportunities or asking for referrals. I tweeted once asking who was hiring and got hundreds of responses and retweets.

Facebook Groups - Oddly, Facebook groups can be a robust source for finding jobs.

Algorithm resources

One great algorithm resource is from John Washam. He's a developer at Amazon and put together an amazing GitHub repository on getting a job at one of the big tech companies.

Best overall resources for interviews

Programming Interviews Exposed - One of the best books on how you can prepare for a programming interview

Cracking the Coding Interview - The classic book for coding interviews

Mastering the Software Engineering interview - A free course from coursera on how to master the coding interview

Geeks for Geeks tutorials - A long list of different algorithms and data structures with explanations and practice problems

Teach Yourself CS - I'm obsessed with this resource. It has an incredible list of many amazing CS resources. I own many of the books on this list.

Coding Interview University - Another truly fantastic resource. This is the most comprehensive list to prepare for a coding interview.

Tech Interview Handbook - An overall guide to some basic tech interviews

Best sites to find practice problems

Project Euler - A list of questions that you can use to make your whiteboarding list

Interview Cake - A resource that walks you through how to solve a huge list of basic questions and the thought process behind the solution

Codewars - A practice problem website where you can find whiteboarding questions

Hackerrank - A great resource to find coding challenge questions

Leetcode - Even more resources for whiteboarding questions

Geeks for Geeks - Different lists of must-do engineering problems

Codewars - Online coding challenges that are language specific

Mock interviews

Pramp - A free resource where peers can interview you

Interviewing.io - Another free resource where you can get mock interviews

Any questions for us?

Reverse Interview Repo - This is an awesome resource with a huge list of questions you might want to ask a potential company.

Frontend interview resources

Here's is a resource on potential frontend developer questions you might be asked in an interview.

Frontend masters also offers a robust free frontend handbook of all you need to know about frontend interviewing topics.

Data Structures & Algorithms in JavaScript - An awesome course by Kyle Shevlin

Stephen Grider's Algorithm and Data Structures course - A good introduction from Stephen Grider