



— The —

WEB DEVELOPER CAREER GUIDE

How to stay relevant, hack
the hiring process, and make more
money as an in-house developer.

CHRIS FERDINANDI

The Web Developer Career Guide

How to stay relevant, hack the hiring process, and make more money as an in-house developer.

By Chris Ferdinandi

Go Make Things, LLC

v1.0.3

Copyright 2016 Chris Ferdinandi, All Rights Reserved

Intro

As an industry, we spend a lot of time talking about innovative CSS techniques, the latest JavaScript frameworks, and new HTML elements. We also spend a fair bit of time talking about freelancing, how to market our services, interact with clients, and price our work.

We don't spend a lot of time talking about being employed.

We don't talk about how to write a resume.

No one ever tells you how to answer the question, "So, what are you making now?" (Don't. It's a trap designed to screw you out of money when it comes time to negotiate salary.)

We don't talk about how "fun" workplaces are complete bullshit, stock options are worse than lottery tickets, or how on earth to have a 5-year plan in an industry that changes every week.

This stuff is insanely important, but we never talk about it. I want to change that.

Before I was a web developer, I was an HR guy at a \$55 billion tech company.

I used to teach engineers and developers how to find new jobs and grow their careers. A few years ago, I used my own advice to change careers, become a front-end developer, work 100% remotely, and grow my salary by 44.8%.

In this book, I share my inside knowledge on how to stay relevant, hack the hiring process, and grow your salary.

There's no such thing as an overnight success.

Having a kick-ass career is really about doing three things over and over again:

1. Talk to lots of people.
2. Never stop learning.
3. Focus on solving problems.

Let's get to it!

How to figure out what to do next

The most common question I get asked is, “How do I figure out what to do next?”

If you have a sense of what you’re interested in (even vague), I always recommend reaching out to some actual people who do that job and see if you can chat with them a bit. Remember this phrase:

Can I learn more about what you do over coffee, my treat?

Some people haven’t responded, but I’ve never had someone say no.

I’ve used some form of this question to chat with the CTO at a \$55 billion tech company I used to work at, as well as some of the big names in web development and people I looked up to.

It’s such a fantastic way to learn what the day-to-day of a role is actually like, what sorts of skills you *really* need to be successful in it, and the typical path someone might take to get there.

Side benefit: if you decide you might want to do that kind of work at some point, you now know someone who can tip you off to openings or introduce you to other people.

What questions should you ask?

Anything you might be interested in knowing, but here are a few to get

you started:

1. How did you end up in this role/doing this kind of work?
2. What does a typical day look like for you?
3. What kinds of skills or experience does someone need to be successful in this role?
4. What industry changes do you see transforming the way this job is done?

I also recommend ending a conversation like that with, “Is there anyone else you think it might be good for me to talk to to learn more?” Keep the cycle going.

This approach works remotely, too. Coffee is a nice in-person ice breaker, but you could chat on Google Hangouts or Skype with people who don’t live near you.

People *love* to talk about themselves. This is surprisingly easy to do.

What if you have no clue what you want to do next?

If you have no clue at all what you want to do next, that’s a bit harder. The blessing and curse of the web industry is just how much it changes and how many options we have.

If you’re in head-scratching mode, you might want to jump start things with a bit of research.

A really easy way to do this is to pick a big conference that you find interesting, dig through the speaker list, and look at the types of places where they work and the things they do there. When you find a few that are interesting to you, do a bit of Googling to learn more about what they do.

If their work seems interesting, I would strongly recommended reaching out and using the “coffee conversation” technique to have a conversation with them.

The myth of the “Five Year Plan.”

Whenever someone asks me, “What’s your five year plan?” I tell them, “I don’t have one.”

In our industry, a five year plan makes no sense. Things change far too quickly. And in all likelihood, your interests will, too.

At one point, I thought I wanted to manage others. Then I was responsible for a team of interns and discovered that I disliked everything about it.

Five years ago, I thought I wanted to primarily be a designer, and learned a little code only to bring my designs to life. Now I consider myself a developer with an appreciation for design, and prefer to spend my time in a text editor or browser.

So how do you know what to focus on?

Pick the thing you think you want to do next—whether that’s a particular programming language or framework, or a type of role—and focus on just that for a month or two.

Dig deep. Learn as much as you can. See where it takes you.

A very successful engineer-turned-VP at a company I used to work for referred to this as “Career Drift.” Even very successful people I talked to who had planned their careers in advance found that their plans changed or fell apart over time. You just never know where something will take you.

Embrace that. Lean in.

And perhaps most importantly, don’t be afraid to quit quickly. Not your job, per se, but the thing you thought you wanted to do next. Throw yourself into it for a month, and if you don’t enjoy it, move on.

Life’s too short. Play to your strengths.

How to keep up

Once you're a more seasoned developer, you start to feel like you can't keep up with the pace of change in our industry.

It's driven in part by a feeling that, as an expert, you should know at least a little about everything. There's also a fear of missing out (sometimes called FOMO)—this fear that everyone will embrace a new thing and you won't and suddenly you'll be the dinosaur still using tables to design website layouts.

This is 100% normal, and you're not alone. Everyone goes through this.

Don't expect your employer to tell you.

I've worked at a few organizations that have amazing learning cultures.

One company had weekly lunch-and-learns, where developers who worked there would talk about cool things they were doing or something new that they learned. We went to conferences. We had memberships to online learning sites.

But I wasn't told what I should focus on, or what I should learn next.

No one knows what you want for your career other than you. Make sure you're taking charge of your own learning and development. Fortunately, there are so many ways to do that as a web developer.

Be a fast follower.

Don't be the first to learn every new technology. Don't jump on every bandwagon. You'll go insane trying to keep up.

Be a fast follower.

Wait until something shows clear signs of sticking around, and then learn it—if you're interested.

Old technology isn't bad technology.

In fact, most businesses would rather work with old, stable, wildly mundane technology than the latest framework.

Static site generators were supposed to replace CMS's like WordPress. Sass was supposed to replace CSS. Grunt was the ultimate build tool, and then Gulp showed up.

The thing is, plenty of companies still use vanilla CSS. Many invested in Grunt workflows, and aren't going to replace them just because they're not stylish anymore. WordPress powers over a quarter of the web (and growing).

Old technology is stable. It's less expensive. It's doesn't change as frequently or break as often. It's easier to find people to work on it.

Don't feel like you always have to chase the latest thing. You can be happily employed working on more mundane technology stacks.

Pay attention to the big shifts in the industry.

Long after responsive web design was considered a best practice, I would still run into more seasoned developers who'd heard of it but weren't really sure how it worked.

This is a really easy trap to fall into when you work as an in-house developer—especially at a larger organization. You may have a relatively stable technology stack, a pretty successful site, or even just a lot of organizational inertia and a culture that doesn't foster innovation.

I was once interviewed by the head of a UX organization who, when asked about their mobile strategy, told me, “I think mobile is a fad, and one that's time is almost up. People don't want to use [our product] on their phone.”

Be aware of big shifts in the industry and how they affect the work you do and the problems you solve.

You don't want to be the one designing layouts in tables when everyone else has moved on to responsive grids and flexbox. That's a great way to make yourself obsolete.

How do you know what skills to focus on?

If you're having [coffee conversations](#), you'll hopefully have a good understanding of what skills you need to be successful in the job you think

you want next.

You can also do a search for openings for jobs you may want in the future, and dig through the job description to find out what kinds of skills and experience they're looking for.

You only need 15 minutes a day.

If you spend just 15 minutes a day learning something new, you'll spend over a week and a half growing your skills over the course of a year (and that's only including weekdays).

So... what can you do in just 15 minutes?

Open source

This how I learned JavaScript.

I decided I was going to figure out how to write a simple accordion script using vanilla JS instead of jQuery.

I Googled. A lot. I tried things. Some worked. Some didn't. Some worked in unexpected ways.

Eventually I had something somewhat decent. I put it up on GitHub (which I had to learn how to do—another new skill).

Then I shared it like crazy. I wrote about it on my blog. I shared it on Twitter. I got questions from people, and feedback. In answering and addressing them, I learned even more.

Then one day, Todd Motto¹ issued a massive pull request for one of my projects, with tons of changes and comments. I learned *so* much from having him rip apart my code.

In retrospect, I would have publicly asked for a code review way sooner.

My point is, you can work on something slowly over time, put it out there, and let the community help you learn. This willingness to teach each other is, in my opinion, one of the best things about our industry.

Online learning

There are a ton of fantastic, interactive lessons on sites like Team Treehouse², Code School³, and Code Academy⁴.

There are also a lot of very talented people sharing great tutorials on their own websites.

Most of the lessons are short. Spend 15 minutes a day, and after a month, you'll have a really solid foundation you can build on.

Communicating your ideas

Don't just focus on technology skills.

I've found the thing that often separates junior and senior developers is not their technology skills, but their communication skills. Being able to clearly document code and explain to others how and why something works is a critical skill if you want to move into a more senior role.

Here's a short list of things you can do to build your communication skills:

- Share code and things you're learning on your blog.
- Give a talk at a local developer event. MeetUp⁵ is a great place to find these.
- Start a podcast (video or audio).
- Hold an internal lunch-and-learn with other developers at your company.
- Talk at a local Small Business Owners Association event. They're always looking for speakers.

A wonderful side benefit of doing these things is that you can add them to your resume and portfolio. They also make your current employer look good, so they're likely to be supportive as well.

What about conferences?

Conferences are great. You'll often come away feeling energized and excited in a way that online learning can't replicate.

They're also a fantastic way to network. I struggle with this a bit. I'm a social introvert. I like talking, but I find large groups overwhelming. I really have to force myself to jump into conversations at conferences. But it's totally worth it. I actually landed a job that way, once.

On the other hand, conferences are expensive. And after a while, they can start to all feel the same.

If you can convince your manager to pay for you to attend, I'd recommend going to one. But if not, don't feel like you're missing out too much.

Pro tip: if your manager won't pay for you to go, you can also apply to be a speaker or volunteer in exchange for a free ticket.

How to find open roles

There are three ways you can find open roles:

1. Search the job boards and submit your resume.
2. Update your LinkedIn profile and get flooded with irrelevant roles from recruiters.
3. Network.

Let's talk about all three.

Using job boards.

This is the most common approach: you search on a job board, find roles you like, and submit your resume.

It can totally work. I got a job at \$55 billion tech company doing that, and it eventually kicked-started my web developer career. But it's *really* hard to do (for reasons I'll cover in [a later chapter](#)).

Let recruiters come to you.

If you've got a little bit of experience and any sort of web presence, you'll eventually start to get emails from recruiters about these “amazing roles” that you're “the perfect fit” for.

This is all a lie.

Many tech recruiters specialize in sourcing candidates, not technology. As a result, they don't have a good understanding of the technology they're recruiting for.

I frequently get emails for Java roles because of my experience working with JavaScript.

Sometimes a company will hire several recruiting agencies for the same role. You might get emails about the same opening from two different recruiting firms (or from two recruiters at the same firm—awkward).

Because of this competition, they typically won't tell you who they're recruiting for. They're afraid you'll just go directly to the company and cut them out of a commission, but it means you have no way to properly evaluate the role.

As a rule, I refuse to work with recruiters who won't disclose the name of the company. Many will try to get you on a phone call before they'll tell you. It's up to you how difficult you want to be, but I insist on a company name before I'll even jump on a call.

None of this means you should never work with recruiters who contact you. But be aware of how they work, how they get paid, and the limits of what they know.

They may seem like a golden ticket, but typically are not.

Network.

This is how I've landed every job I've ever had, with one exception. Networking is **the** best way to find a job, for two reasons.

First, hiring someone is always a gamble. Hiring someone that a person who already works for you likes and trusts is less of a gamble. You come pre-approved.

Second, the process of getting a position approved can be slow. Like, really slow. Most hiring managers don't want to wait, so they start networking with people to find candidates before the job is even approved or posted anywhere. By the time you see a role on a job board, there may already be a short list of final candidates.

Networking helps you find out about these roles sooner and get in before the door closes.

What's the best way to network when you're looking for a job?

Remember this phrase?

Can I learn more about what you do over coffee, my treat?

It works when looking for a job, too—even if you already know what the role entails and have the skills to do it.

It's just so much easier for someone to say "yes" to than, "I'm looking for a new job. Can I chat with you about openings at your company?"

It's like a magic question that gets people to open up and let their guard down.

The one difference with this approach when you're actively looking for work: ask if they know of any open roles that might be a good fit for you. If you've never done this before, it's amazing how willing people will be to help you out.

It can be a slow process, which is why it's good to start now and not when you're *actually* looking for a job.

But it works! I've landed jobs through old coworkers, someone I met at a conference, and by using coffee conversations to talk to friends of friends of friends until I eventually found someone who was hiring.

Your online portfolio sucks (and how to fix it)

Most web designer and front-end developer portfolios are pretty basic:

- The name of the client.
- One or two sentences about what you did for them.
- An image or three of the website, logo, or whatever.
- Random keywords about your role. Example: “Branding, Development”.

I’m going to be blunt: these kinds of portfolios suck.

What potential employers really want to know is how you added value for the client. Pretty pictures of your work don’t tell them that.

What you really want is a collection of case studies.

Go ahead and include images of your work. But also talk about the key challenges your client had or the big goals for the project. Talk about the decisions you made, why you made them, and how they impacted your objectives.

What did you do, *why* did you do it, and what was the *outcome*? You should also get quotes from happy clients and include them if you can.

Here's the beginning of a case study I wrote for PAWS New England, a client of mine:⁶

PAWS New England rescues abandoned and abused dogs and places them in safe and loving homes. They have no physical shelter, and rely on a network of foster homes to care for the dogs they rescue prior to adoption. As a result, a strong online presence is critical to their success.

PAWS does amazing things, but their website and social media accounts were not. I partnered with them to redesign their entire digital communication strategy, with powerful results.

The Highlights:

- Doubled annual donations.
- PAWS was featured in an HBO documentary about shelters and rescues.
- Drove an additional \$800k in revenue over a four year period.
- Web traffic is up 4.7x, and mobile traffic is 27x higher.

“What you’ve done for PAWS... the word ‘life changing’ comes to mind.” - Kelly Parker, PAWS New England Cofounder

Why job hunting sucks so bad

Before the internet, if you wanted to apply for a job you had to browse through job listing in the local paper and physically mail in your resume. It was time-consuming and friction-filled, and as a result, companies received a pretty manageable amount of applicants for jobs.

The web makes it insanely easy to find and apply for jobs. You can search by keyword for jobs anywhere in the world, and then apply with just a few clicks.

Originally, this was awesome.

You could easily find the perfect job, and employers could access a larger pool of applicants more easily.

And then the economy got bad.

Suddenly, businesses were getting flooded with hundreds or thousands of resumes.

People were desperate for work. Many weren't really qualified for the roles they were applying for. But the process is so frictionless, they sent their resumes in anyway.

There's no way a recruiter could actually sort through all of those resumes.

So people built software to do it for them.

An Applicant Tracking System (or ATS) is a piece of HR software that collects and tracks resumes.

It's most powerful feature: automatically filtering resumes. Recruiters can provide keywords for each role, and the ATS scans resumes for those keywords and tries to determine how good of a match for the job the person is.

If the ATS doesn't think you're a close enough match, your resume never gets seen by a real person.

Even if you make it through the ATS, a recruiter may only look at your resume for a few seconds.

Depending on the role and the company, hundreds of resumes might still make it through the ATS. The role you're applying for may be one of dozens that the recruiter is responsible for.

One study found that your resume will only get looked at for six seconds.⁷

Your resume has an insanely difficult task.


First, it has to make it past the ATS and get seen by a real person. Then it has to convince that person—in just six seconds—to keep reading instead of putting it back in the stack.

Here's the good news: now that you know how this process works, you can design for it.

In [the following chapter](#), I'm going to teach you how to create a resume that's more likely to make it past the ATS, and more likely to break past the six second mark.

Your resume sucks (and how to fix it)

There's a good chance that you have an "Objective" section up at the top of your resume. It probably says something like this:



Seeking a role as a front-end developer where I can utilize my talents and skills to fulfill the goals of the organization.

What you're really saying is, "I'm looking for whatever job you have available."

It's the same thing that everyone else puts on their resume, it's the first thing that the recruiter sees, and it tells them nothing about why they should keep reading.

Replace your objective with a "Summary of Qualifications."

A Summary of Qualifications is a short blurb that summarizes why you're such a kick-ass developer.

Ideally, it's one or two sentences followed by a short bulleted list of skills. Here's an example:

A front-end developer specializing in responsive web design. Skills and experience include:

- Mobile-First RWD
- HTML and CSS
- Native JavaScript and jQuery
- QUnit and Jasmine
- Sass & SCSS
- Gulp, Git, and Travis CI
- Web Performance
- WordPress and PHP

A Summary of Qualifications shows recruiters why they should keep reading past the six second mark. Each skill or area of expertise you point out should be supported later in the resume.

Move your education to the end.

Most resume templates have this right up at the top. The thing is, no one cares. Seriously.

(Unless you have a PhD in Computer Science. Then they probably do. But you should still put your education at the end.)

What you know is important, but what you've done with what you know matters so much more. You want to get to your experience as quickly as possible.

Focus on outcomes, not responsibilities.

It's pretty common to see things like this under job experience on a resume:

- Know HTML5, CSS3, and mobile-first, RWD.
- Experience in accessibility and progressive enhancement.
- Responsible for unit testing and QA.
- Created responsive websites for clients.

So what? Everyone else who applied for this role has done those things, too.

Tasks make you an expense. Outcomes make you an asset that adds value to the organization.

You want to show the recruiter why those things matter by talking about the value you created for the client or the organization that you were working for. Here are some examples:

Redesigned Animal Rescue Organization's website to be mobile-friendly, resulting in a 100% increase in online donations.

Made performance optimizations to Big University website, decreasing page load time by 500%.

One easy way to remember this: every time you add an experience to your resume, imagine a little bird sitting on your shoulder asking, "So what?"

What if you don't have numbers or data for your experience?

You can still show impact without numbers. For example:

Integrated the MailChimp API into Acme Corp's CMS, reducing the amount of time they spend cross publishing content on their site each week.

Summarize the work you did above your list of specific experiences and outcomes.

For each project or role on your resume, you'll end up with a bulleted list of experiences and their outcomes. Above the list, add a short one or two sentence summary of your work. Here's an example:

Animal Rescue Organization rescues abandoned and abused dogs and places them in safe, loving homes. I redesigned their entire website to be mobile-friendly.

- Built on a responsive, mobile-first grid, the site scales beautifully from small screens to big ones. Since the redesign, mobile-traffic grew from just 9% of all traffic to 52%.
- Wrote progressively-enhanced JavaScript for modals, expand-and-collapse widgets and drop-down menus, ensuring access to content on devices without JS support.
- Focused on performance as a design criteria, resulting in average page load times of just 1.5 seconds and minimal HTTP requests.
- Used SVG icons instead of raster images, resulting in lightweight, scalable icons that look great on high-density displays and can be easily styled using CSS.

Customize your resume for every role you apply to.

Remember, recruiters use an ATS that screens your resume against a list of keywords.

Often times, the recruiter sourcing candidates isn't technical themselves.

As a result, they'll often just pull the keywords they use directly out of the job description. Many times they won't use synonyms or alternative words with similar meaning, simply because they don't know any better.

Now that you know this, you can hack the process.

When describing your experiences, pull your language directly from the job description. Don't get creative.

For example, if the job description says:

Experience with responsive web design

Your resume should say:

Built a website using responsive web design

And not:

Built a website using mobile-first RWD

Don't assume "mobile-first RWD" will match or that the recruiter will know that they're the same.

A caveat: I'm sure this goes without saying, but only list experiences if you actually have them. You might trick the ATS, but eventually the recruiter or hiring manager will figure out that you don't have the experiences you said you did. Recruiters talk. They'll tell others about you.

Your Summary of Qualifications should get customized, too.

If I was applying to a role that talked about using JavaScript frameworks, I'd use something like this:

A front-end developer specializing in custom JavaScript development. Skills and experience include:

- Native JavaScript and jQuery
- QUnit and Jasmine
- Gulp, Git, and Travis CI
- Web Performance
- Mobile-First RWD
- HTML and CSS
- Sass & SCSS

If I was applying for a different role that was focused on building an internal design system, my Summary might look more like this:

A front-end developer specializing in internal design systems.

Skills and experience include:

- CSS frameworks and styleguides
- HTML and CSS
- Sass & SCSS
- Native JavaScript and jQuery
- QUnit and Jasmine
- Gulp, Git, and Travis CI
- Mobile-First RWD

Remember, the goal of this section is to get the recruiter to read your resume for more than six seconds. You want to make as many direct links to the role they're hiring for as possible.

Keep it short.

You may have heard that resumes should always be one page. That's wrong.

For someone straight out of college, *maybe*, but if you have any amount of experience, it's ok to go on to a second page. You don't have to fill up the entire second page either. It's far more important that you showcase your experiences and how they match the role.

I recommend the "One-to-Three Guideline:"

- Keep each experience and summary to one to three sentences.
- Keep your resume to one to three pages.

What if you have a lot of experience?

I've seen resumes that were (no lie) 12 pages long. These were very experienced engineers.

Recruiters aren't going to read that. And most of the experiences in a resume that long aren't relevant to the job you're applying for, either.


If you've been in technology for a while, or if you've just switched careers from another field (like I did), I'd recommend summarizing older or irrelevant experiences. You can do this by simply dropping the bullet points.

So this:

Animal Rescue Organization rescues abandoned and abused dogs and places them in safe, loving homes. I redesigned their entire website to be mobile-friendly.

- Built on a responsive, mobile-first grid, the site scales beautifully from small screens to big ones. Since the redesign, mobile-traffic grew from just 9% of all traffic to 52%.
- Wrote progressively-enhanced JavaScript for modals, expand-and-collapse widgets and drop-down menus, ensuring access to content on devices without JS support.
- Focused on performance as a design criteria, resulting in average page load times of just 1.5 seconds and minimal HTTP requests.
- Used SVG icons instead of raster images, resulting in lightweight, scalable icons that look great on high-density displays and can be easily styled using CSS.

Becomes this:



Animal Rescue Organization rescues abandoned and abused dogs and places them in safe, loving homes. I redesigned their entire website to be mobile-friendly.

Small details that make a big difference.

These may seem stupid or unimportant, but when a recruiter gets hundreds of resumes, anything they can use as a filter helps them reduce the number of candidates they have to evaluate.

- Run spell check on your resume. Always.
- Have someone proofread your resume. Don't do it yourself. You'll miss things, including grammar quirks and words that are real words but not the one you meant (your/you're).
- Use past tense, even for current roles. It makes the resume read more consistently.

How to kick ass at your next job interview

During an interview, employers are trying to understand two things:

1. Do you have the skills and knowledge to do the job?
2. Are you someone they could actually work with every day?

In this chapter, I'll show you how to do a better job with both of these.

What types of questions will you get asked?

For most jobs, there are three types of questions an interviewer will ask:

1. **Yes/No.** Questions that can be answered with a “yes” or “no” response (more on this in a minute). Example: “Do you have experience with Sass?”
2. **Situational.** Questions that ask you how you would handle a hypothetical situation. Example: “What would you do if you needed to complete a project with a JavaScript framework you’d never used before?”
3. **Behavioral.** Questions that ask you to explain how you handled a real life situation you’ve been in. Example: “Tell me about a time when you had to complete a project using a new technology.”

The best way to answer all three of these types of questions is with the STAR Method.

The STAR Method

The STAR Method is an acronym:

Situation

Task

Action

Result

To answer a question using this method, identify a Situation or Task you were in, describe the Actions you took, and then explain the Results you achieved.

Remember, tasks make you an expense. Outcomes make you an asset that adds value to the organization.

Use this format even for Yes/No questions. It allows the interviewer to learn more about how you work and solve problems, and demonstrates your ability to add value.

Let's look at some examples.

“Tell me about yourself.”

A common answer:

I'm from Ohio but I went to school in Boston. I graduated with a degree in communications, and have been working at Animal Rescue Organization as a web developer.

I like sports and my favorite food is pasta. I'd like to continue working as a developer.

Here's a better answer:

I'm a front-end developer who specializes in responsive web design. For the last two years, I've been working at Animal Rescue Organization as a web developer, where I help manage their web presence.

I was part of a redesign initiative to make their site mobile-friendly. As a result of the redesign, they doubled their annual donation revenue.

The first answer does nothing to sell yourself as a value-adding asset to the company. The second immediately showcases both your skill (RWD) and it's impact (doubling annual donations).

“Do you have any experience with Sass?”

A common answer:

Yes.

A better answer:

Yes. In my role as a front-end developer at Animal Rescue Organization, I used Sass as part of our mobile-friendly website redesign project.

I used variables and mixins to speed up our development process and ensure consistency in design. Nesting and small, modular files allowed us to keep our project organized while managing a large stylesheet.

Again, you can see you're identifying a situation or task, talking about the actions you took, and identifying the results.

This is a great response if you have the experience they're asking about, but what if the answer is "no?" In that case, you want to answer honestly and then talk about another situation in which you quickly learned a new skill.

I haven't worked with Sass, but I'm quite comfortable learning new technologies. Before becoming a front-end developer, I was an HR professional. By using online tutorials like CodeSchool and building a network of mentors, I was able to teach myself web development very quickly.

Most recently, I learned how write vanilla JavaScript instead of relying on jQuery. As a result, my scripts have one less external dependency and load faster.

Technical interviews

For web developer roles, there's a fourth type of interview question you're likely to encounter: the technical interview.

These are a form of quiz, assignment, or exercise designed to gauge your technical ability in a more meaningful way. The problem is, many of them are bullshit.

Technical interviews typically involve answering very detailed, academic questions on-the-spot, or coding on a whiteboard or piece of paper while people watch you.

If you're a good test taker, you may actually do quite well at these. But I'm not. And they bother me, because they're not at all what coding in real life looks like.

When I write code, no one is staring over my shoulder asking me why I did something.

My code doesn't always run right the first time. I check it in a real browser, inspect things in developer tools, and use Google to figure out something I'm stuck on.

That is totally normal.

Technical interviews create artificial pressure that by design cause people who aren't good at tests to do poorly. I'm not alone on this.

Ike Ellis explains:⁸

Sometimes they would want to hear something I didn't know. Other times I just froze on topics that I know very well. (One time I couldn't even name my favorite video games.) Many times, I failed to perform well on some logic puzzle. Every job I actually got was because a friend made sure that it happened. For years and years I lived in fear of the interview because I knew that I'd fail.

There's a better way to demonstrate your technical skills.

The trial assignment.

This is an approach used by companies like Automattic (the WordPress people)⁹ and Harvest (the time tracking software).

You're given a coding assignment—based on the kind of work you'd actually be doing—and then go off and complete it within a predefined amount of time (anywhere from a few days to a couple of weeks).

So what do you do if you're asked to come in for a technical interview?

You have three choices:

1. You say “ok” and go to the interview. If you're good at taking tests or working under pressure, this may work out just fine for you.
2. You politely ask if they would be willing to give you a coding assignment instead. Explain that you know you've got the skills to do

the job, but that technical interviews don't best allow you to show off your talent, and ask if they can give you a mini project to work on instead. I've never had the guts to try this.

3. Decline the interview.

These days, I would probably go for option 2, and if that doesn't work out, 3. However, early in your career, you may have to suffer through some technical interviews (or totally nail them) to get the job you want.

How do you prepare for an interview?

There's a few simple things you should do:

1. **Study your own resume.** You want to know your own background (and outcomes) inside and out. I find myself referring back to two or three examples that best highlight the value I add over and over again.
2. **Research the organization.** Make sure you know their products or services, how they make money, and any big accomplishments they've had recently.
3. **Take notes.** You'll appear more interested and engaged. But don't take so many that you stop paying attention to the interviewer. I use notes in two ways:
 - To write down questions I have so that I don't forget them and can ask them at the end when they say, "So, do you have any questions for me?"
 - As a side benefit, I cheat and write down two or three of my best work examples in the corner of the page so that I don't get

nervous, draw a blank, and completely forget them.

4. **Practice.** It's super awkward, but get a friend to ask you common interview questions and practice answering them with the STAR Method. You want to have this down cold when you go in for an actual interview.

About 70-percent of communication is nonverbal.

That means that what you say in an interview is important, but so is what you do.

Remember, interviewers aren't just looking to make sure that you have the right technical skills. They want to make sure they'll be comfortable working with you every day.

Chimpanzees and job interviews

I have a degree in Anthrology. One of my favorite anthro classes was primatology, the study of our closest living ancestors and what we can learn about being human from them.

One thing I learned is that when two chimps greet each other and one makes a facial expression, the other will involuntarily mimick that expression.

The reason is because facial expressions and other body language don't just communicate your mood. They influence it.

This happens in humans, too.

Next time you walk by someone, smile.

There's a good chance they'll involuntarily smile back at you. When they do, they'll feel a bit happier. This allows chimps (and humans) to communicate moods and emotions without words.

You can use this to your advantage during interviews:

- **Smile.** The interviewer will enjoy interacting with you more.
- **Sit up straight.** I tend to slouch, and need to constantly remind myself of this. Good posture conveys confidence.
- **Make eye contact.** Don't stare, of course. But eye contact makes you appear confident and friendly.
- **Give a firm handshake.** I've had actual recruiters tell me they didn't hire someone because they had a limp handshake. Is that a bullshit reason not to hire someone? Absolutely. But it actually happens.
- **Dress the part.** For web developers, this thankfully doesn't mean a suit. But you should try to dress one notch up from how people who work there dress. If they're normally in shorts and flip-flops, wear jeans and closed-toe shoes. Not sure what the normal attire is? Call the company and ask the person at the front-desk.
- **Show up early.** I build in a 15 to 20 minute buffer for traffic and unforeseen issues. Don't go in until about five minutes before, though. Even if you tell the front desk not to, they'll call the person you're there for to let them know you've arrived, which can stress them out.
- **Shut off your cellphone.** Seriously. Shut it off. You don't want it ringing mid-interview.

A quick but important note: *These recommendations are based on my experiences in North America. Cultural norms around things like eye contact and handshakes can vary around the globe, so please adapt these based on where you're based and where you're interviewing.*

Be aware of cell-phone lag.

Cell phones have a slight lag from when a person speaks to when you can hear them. This is intensified when the person you're speaking to is also on a cellphone.

If you're using a cellphone for a phone interview, try to wait a moment before speaking.

Otherwise, you may think the interviewer is done asking their question and begin talking, and end up accidentally speaking over them. You may not even realize it, but they'll think you're rude or just generally unaware of others.

How to get paid what you deserve

One area of the recruiting process that can be a real mystery is how to negotiate salary.

What if you ask for too much and they decline? What if you ask for too little and screw yourself out of money? How do you know what a reasonable salary is for the role?

How do you know how much to ask for?

Salary varies wildly based on:

- The type of work involved.
- The size of your company.
- Where they're located.
- Whether they're a product company or an agency.

Fortunately, there are two tools you can use to get a ballpark:

1. Glassdoor's Salary Calculator^{[10](#)}
2. Indeed^{[11](#)}

At Glassdoor, you just enter the job title and location and it will provide you with a salary range. At Indeed, you can search for roles by location. Many list a salary range that you can use as a baseline.

Pick a number in the middle of the range if you want to be conservative, or

towards the higher end if you have a rare or unique skill that you think is worth top dollar.

Agency versus In-House Developer

A web agency—one who makes web things for other companies—typically pays less than a business that sells products or sells services that are not web development (but where you happen to be a developer).

This is a trade-off you'll have to make.

Agencies can expose you to tons of different companies and development teams, but doing high-volume work for lower pay. Businesses can often pay you much better, but with less variety.

There's no right choice. Just different options.

Variable versus Fixed Compensation

There are two types of monetary compensation (ie. money):

1. Fixed
2. Variable

Fixed compensation is a guaranteed salary (for example, \$65,000 a year).

Variable compensation is typically a bonus whose amount can change based on things like individual, team, or company performance.

You may, for example, be offered a \$5,000 annual bonus target. If the company hits its estimated numbers, you'll get 100% of that. If they don't hit their numbers, you may only get 85%, or none of it.

Variable compensation can be large and enticing. I personally always prefer a higher fixed compensation, since it's guaranteed money in my check every week.

Perks

Companies will often offer perks to offset the salary they offer you. Some are better than others.

Good

- Health care, which is really expensive if you pay for it yourself.
- Professional development, like conferences, memberships, and in-person training.
- Paid time off, which costs them little but is awesome for you.

Bad

- Stock options, which are like lottery tickets that you have to work harder for.
- "A fun work place," which is often used to make you work harder and longer for less money.

Location-hacking for more money

Many web developer roles today are remote ¹². Depending on where you live, you can take advantage of this to make more money than you would working locally.

For example, San Francisco has the highest cost of living (and salaries) in the US, followed closely by New York City. If you worked in Vermont or Oklahoma, which have a low cost of living, you could negotiate a salary that's cheaper than what they might pay local developers, but higher than what you'd make where you live.

You'll probably have to travel a few times a year, but working from home and making more money are a pretty sweet trade-off for a few weeks of travel.

How do you ask for your desired amount?

When is almost as important as *how*.

Most recruiters like to ask up front, and it makes sense. If your requirements are way above the salary range for the job, there's no sense in moving forward. That's one strategy, and it's totally valid.

Another, equally valid approach is to delay all salary discussions until they're ready to make you an offer. By this point, they're ideally already very excited to have you join, and may be willing to push their budget a bit to make it work.

So *how* do you ask? Just tell them what you want to make, calmly and confidently.

I'm looking to make \$65,000.

Practice saying that phrase over and over again.

You want to be confident when you say it. It should sound so second nature that you've never doubted it or considered that it could be too high.

You should be comfortable asking for money. After all, it's why you work, and your employer would have no problem paying you less if you let them.

What to do when they make an offer

There's a bit of a dance that happens here.

If they offer exactly what you asked for...

Sometimes, they'll come back right away with exactly what you asked for. That likely means that the amount you asked for was too low, but you can't up your salary requirement at this point. Still, you got the offer, and for the amount you wanted. Congrats!

(Ask for more next time.)

If they offer you a bit less than you asked for...

Other times, they'll come back with something a bit lower and offer some benefits and other non-monetary compensation. This means they're really interested, but are trying to get you for less money.

Always counter offer, but your personal situation will dictate by how much.

For example, if you think the role would be a huge boost to your career, or their health care package is really good, it might make sense to compromise a bit. If you asked for \$65,000, and they offered you \$60,000, you might say:

Make it \$63,500 and we have a deal.

There may also be a perk you care about more than money, like paid travel to a conference, or an extra week of vacation.

Make it \$62,000 and throw in an extra week of vacation, and we have a deal.

If a higher salary is your number one requirement, you might prefer to remain firm. You might say:

While I certainly appreciate all of the wonderful benefits you have, my salary requirement is \$65,000. If you can offer me that, I'll accept the job today.

If they low-ball you...

Sometimes, the recruiter will appear taken aback by your amount. “The top of our range for this role is \$55,000. Our more senior developers don’t make what you’re asking.”

Side note: *a version of this conversation has actually happened to me.*

It’s likely that they’re telling you the truth. If you’ve done your homework and know that what you’re asking for is a fair price, the economics probably just don’t work for their business.

You might say, “I completely appreciate that. Unfortunately, my salary requirement is \$65,000, so I don’t think this is a good fit. Best of luck in your search!”

Always be polite. Just because they’re not hiring you now doesn’t mean you won’t cross paths again in the future.

“How much do you make in your current role?”

Recruiters will often ask this question. It’s even on some job applications.

Never, ever, ever answer this question. It’s a trap¹³ designed to screw you out of money.

What you make today should never be a basis for what you’re paid at your next job. Your pay should be based on what others in similar roles in similar locations are paid, and on the value that you add to the company.

So how do you answer this question if you’re asked? By calmly and confidently telling them what you’d like to make:

I'm looking to make \$65,000.

I've never had a recruiter repeat the question after that response, though it could happen. I'd just repeat my answer.

What now?

If you remember nothing else from this book, I hope you'll remember these three key takeaways:

1. **Talk to lots of people.** “Can I learn more about what you do over coffee, my treat?”
2. **Never stop learning.** Pick one thing to learn, dive deep, and quit quickly.
3. **Focus on solving problems.** Tasks make you an expense. Outcomes make you an asset that adds value to the organization.

If you have any questions or feedback, or just want to say hi, email me at chris@gomakethings.com.

About the Author



Hi, I'm Chris Ferdinandi. I help web developers have kick-ass careers.

I love pirates, puppies, and Pixar movies, and live near horse farms in rural Massachusetts. I run Go Make Things with Bailey Puppy, a lab-mix from Tennessee.

You can find me:

- On my website at [GoMakeThings.com](https://gomakethings.com).
- By email at chris@gomakethings.com.
- On Twitter at [@ChrisFerdinandi](https://twitter.com/ChrisFerdinandi).

-
1. <https://toddmotto.com>↵
 2. <https://teamtreehouse.com>↵
 3. <https://www.codeschool.com>↵
 4. You <https://www.codecademy.com>↵
 5. <https://www.meetup.com>↵
 6. <https://gomakethings.com/projects/paws-new-england/>↵
 7. <http://cdn.theladders.net/static/images/basicSite/pdfs/TheLadders-EyeTracking-StudyC2.pdf>↵
 8. <https://medium.com/ikeellis/i-will-not-do-your-tech-interview-80ba19c55883>↵
 9. <http://davemart.in/remote-hiring/>↵
 10. <https://www.glassdoor.com/Salaries/index.htm>↵
 11. <http://www.indeed.com>↵
 12. <https://weworkremotely.com>↵
 13. <http://i.giphy.com/3ornka9rAaKRA2Rkac.gif>↵