How to be an uncommonly good web developer?

Jessica Oei

In-depth web development tutorials, productivity and career advice.

https://coder-coder.com/subscribe

Dec 23

What does it take to be a really good web developer?

If you’re working at your first programming job, you probably found out quickly that it’s not easy.

It’s one thing to watch coding tutorials, read programming books, and make portfolio sites. It’s quite another to have to build websites from spec, to meet deadlines, and most importantly, to make sure that your bosses and clients are happy!

On top of all that, technology changes fast. You may feel like you have to stay on top of trends or risk becoming obsolete next year.

最重要的是，技术日新月异。你不得不紧跟技术潮流，否则就有可能落伍。

**Want to know a secret, though? You don’t have to be afraid.**

想知道一个秘密吗？ 你不必害怕。

As you keep working, you will gain experience in your job. In the same way that you’ve learned coding, you can also learn how to perform at a high level at work.

随着你持续工作，你就会从你工作中获取经验。用同样的方式，对于程序员工作，你可以在工作中学习怎样表现的更好。

It’s possible to learn how to be a good web developer. And it’s even possible to be an **indispensably good web developer**. (Obviously I can’t guarantee your job security, but you know what I mean.)

学习成为一个好的web开发者，不是难于上青天的。甚至，也有可能成为一个不可或缺的开发工程师。

**My first job**

I’m a self-taught web developer, who never took a formal computer science course. I’ve currently been in the field for about seven years. And of course, I’ve struggled with the learning curve and with impostor syndrome.

But I’ve also picked up a lot of valuable experience and skills along the way. You might find some similarities to my experience and your own.

When I started out as a junior web dev in my first real job, I was beyond thrilled.

But to be completely honest, I was terrified for the first two years.

Every day I would have to do something for work that I had no idea how to accomplish. And I often didn’t have a clue how to even get started!

I felt incompetent a lot of the time.

Thankfully though, everything always worked out. I had really supportive supervisors and coworkers who showed me the ropes. And I somehow met all my deadlines.

Over time, I wasn’t feeling as stressed as before.

And after four or five years, I actually started feeling confident in my skills. I know five years may sound like a really long time to you. Especially if you’re just starting out.

Just remember this:

**Success will never happen overnight. But it often happens after days, weeks, months, and years of trying hard and not giving up.**

I feel like I’ve learned a lot in my own brief career. And I want to help newbie web developers who have fought so hard to land their current jobs.

There are some principles that I think will help you to stand out in a crowd. Applying them can enable you to become an asset to your company and your customers.

And even if you’re not working yet, you can still apply most of these principles to your learning process.

Want to become an uncommonly good web developer? Read on!

**#1: Understand that mastering anything is a marathon, not a sprint.**

If you’ve landed a job straight out of bootcamp or college, that’s great! Just know that you’re at the beginning of your coding journey.

Web development is a broad field, and like any skill, it takes a long time to really get good at it. You have a long road ahead of you, and at this moment, there is a lot that you don’t know.

But that’s ok! You’re a beginner. You can’t be expected to be an expert in everything right now.

Don’t expect that you have to know everything there is to know, right this second. It’s simply unrealistic, and it will cause you a lot of stress.

Obviously you should want to learn and get better at web development. **But try to keep a bird’s eye view of your progress and be realistic about where you are right now.**

Think in the long-term sense of your career, not just in the short term of this current moment. And don’t give up too early.

**#2: Try as hard as you can first before asking for help.**

In those early years, the feeling of having to explore uncharted territory was anxiety-provoking. I would feel this pressure of having to figure something out that I just didn’t know.

**Knowing when to ask for help**

In that situation, I had two choices. I could go back to my boss and say, “Sorry, I don’t know how to do this. Can you do it for me?”

That might have taken the pressure off in the short-term, but it wouldn’t have been good for me in the long run. Because an employee that doesn’t know how to find solutions will take time away from everyone else.

The other choice would be to first do as much research as possible on my own. Eventually I would hit the end of what I could understand.

Then I would finally go to my boss and say, “Sorry, I can’t figure this out. But here’s what I’ve tried so far, and I think it might be related to this…”

See the difference?

**Instead of giving up early and asking someone else to do my work for me, I did as much as I could myself.**

And when asking for help, I showed that I had done some work on my own.

**Learning how to teach yourself**

Researching and testing on your own will help you learn how to teach yourself new concepts. This benefits you in two ways:

You’re learning new skills on the job — essentially, you’re getting paid to learn. You’re not sucking time away from your boss or coworkers with questions that you could figure out yourself. You’re an asset to your company, not a detriment.

Of course, the other extreme is that you spend several hours or even days stuck on a problem when you could have gotten the answer quickly by asking for help. That’s not good either because at that point, you’re spending too much time on the problem.

You’ll have to figure out for yourself how much to try on your own, and when to ask for help. But through experience, you’ll be able to find a happy medium.

**#3: Stay curious.**

When I first started my job, I frequently got positive feedback that I soaked up information like a sponge. I was willing to try anything, even if I didn’t know exactly how to do it. And I was honestly curious to learn how things worked.

**This desire to understand how things work is at the heart of every engineer. Let your curiosity feed that desire.**

Don’t shut it off just to get things done. Understanding the *why*is extremely important.

If you’re researching how to do something, don’t just stop once you’ve found the answer in some Stack Overflow code snippet. If you’re not pressed by an urgent deadline, take a few extra minutes to look at the solution and how it fits in with your code.

Try to understand why your code wasn’t working before, and why it is now. It’s ok if you don’t understand everything 100%. Just try to grasp the concept. Even if you don’t totally get it this time, you’ll be better equipped to further your understanding next time you encounter the same problem.

**#4: Always be improving.**

This applies not just to web development, but to any area of life: cooking, driving a car, even relationships.

**Analyze and learn from your mistakes**

When you notice that something doesn’t go quite right, step back and get some perspective. Ask yourself: what did you want to happen, and why didn’t it happen?

If your code broke a website, what happened, and how can you avoid it in the future? Or if you got into an argument with a coworker while trying to explain something, what happened there?

Try to do the following:

* Go back and analyze that non-optimal situation.
* Figure out what happened.
* Think of actionable steps that you can take to avoid the problem next time.

It takes time, and sometimes we truly don’t have that time to go back and analyze every single decision.

But constantly glossing over mistakes and conflicts without learning from them will ensure that you will repeat your mistakes.

It’s tough to do, because when something negative has passed, the natural instinct is to not want to dwell on it.

Think about it like debugging your life. If you keep making the exact same mistake in your code, that would be pretty inefficient. It means that there’s something that you need to learn so that it won’t happen anymore.

Even if it’s just with tiny mistakes, practicing this mixture of self-awareness and self-improvement will reap you rewards in the long run.

**#5: Get really good at solving problems**.

As a web developer, a large chunk of your job is solving problems. Whether you’re writing new code or debugging existing code, the end goal is to make it do what you want.

So how can you get good at solving problems in code?

First, you have to understand exactly what it is supposed to do, and all the parts involved in the process.

Look for clues as to why the problem is happening. Then you’ll be able to solve the mystery!

**Fixing a lamp**

For illustrative purposes, let’s say that you have to fix a lamp that isn’t working.

In order to troubleshoot this lamp, you have to first identify the optimal behavior. In this case, it would be that switching the lamp to “on” will cause the light bulb to turn on.

If this isn’t happening, you could go through and check each step in the process to determine what’s failing.

Here’s a list that you might make:

* **Step 1**: The electricity in the house comes through the outlet. Check that the electricity in the house is on, and check that the outlet is working.
* **Step 2**: The lamp is plugged into the outlet to allow electricity to travel through the lamp. Check that the lamp is plugged in.
* **Step 3**: The electricity travels through the wire and switch in the lamp. Check that the wire and the switch are both working.
* **Step 4**: When the switch is turned on, the electricity reaches the light bulb and turns it on. Check that the light bulb is working and not broken.

**By testing each step, you can use the process of elimination to narrow down and identify the culprit of your problem.**

**Debugging: the best and worst parts of a web dev’s life**

Finding and fixing a problem in your code will be more complicated than testing a light bulb, of course. But the process is the same.

I’m not going to lie, debugging code and troubleshooting issues and errors can be extremely frustrating. There have been times where I’ve spent literally days or weeks stuck on a particular problem. These are the worst parts of being a web developer.

However, the flip side is that when finally discover that solution, you will feel amazing. You might almost feel all-powerful, at least in that moment. That feeling of exultation is without a doubt the best part of being a web developer.

**Don’t overlook the non-technical skills.**

The last two tips will be quite a switch from the topics so far. There are some other aspects of being a professional web developer that have absolutely nothing to do with coding.

You might call them people skills or soft skills.

**People skills are just as important as coding skills**

Even though we work with code, we are ultimately working with people. The websites and apps that we build are meant for humans. And no matter your job, you’ll be working with human customers, clients, or coworkers.

**The best web developers aren’t just good at coding. They are also really good at understanding the big picture, and at working with people.**

I personally think that these last two principles can really help you pull away from the pack. They’re not easy to master because they require a different kind of complex thinking.

And many web developers may not consider the need to develop these non-technical skills. This is where you can begin to make yourself uniquely useful.

**#6: Cultivate empathy towards others.**

Empathy involves being aware of and understanding of the feelings, needs, and perspectives of others. In short, it’s the ability to put yourself in their shoes.

**It’s not about “winning”**

When my husband and I have fights, we start off both feeling at odds with each other. I might feel like I’m 100% right — which is never the case in real life! So we talk everything through, even though it’s painful and difficult to do.

Eventually we realize that it was all based on a misunderstanding or one of us had a bad day. We’re both able to finally see the situation from the point of view of the other person.

It’s not about being right or “winning” the argument. It’s about being empathetic and being on the same page as the other person.

**Being empathetic at work**

In the same way, when you work with people you will inevitably have conflicts or disagreements.

But try to avoid jumping to conclusions and demonizing the other person. Most likely anything they did or said wasn’t meant personally.

**Try to talk to them and find out where they are coming from.**

They might have been really busy and simply forgotten to email you back. They might have just had a horrible meeting and regrettably took their stress out in you in the form of a terse reply.

**Respecting others**

And beyond giving others the benefit of the doubt, empathy at work also involves being aware and respectful of others’ workloads and situations.

Let’s say that one of your coworkers is working like mad on a close deadline. Don’t bother them with unrelated issues that really aren’t urgent while they’re in crunch time.

But it might be a nice gesture to ask them how it went once things have calmed down.

On a related note, if you’re interested in reading more about workplace etiquette, I’d recommend checking out the advice blog [Ask a Manager](http://www.askamanager.org/). It’s super informative about what to do and not to do in the workplace.

**#7: Be proactive, not reactive.**

In my opinion, being proactive versus reactive is a quality that differentiates successful people from mediocre people.

**Being reactive**

A reactive employee will do the work that is asked of them, but they will do no more than that. If something goes wrong they will fix it, but they won’t think ahead to prevent this problem from recurring.

They don’t consider the big picture or how interlocking parts of a system work together. The reactive person’s default mindset is: “If it doesn’t affect me, it’s not my problem.”

It’s very short-term thinking. Note that they are not doing bad work, but they are missing out on a lot that they could be doing.

**Being proactive**

In contrast, the proactive employee won’t just complete their assigned work and call it a day. They will also make small improvements as they go.

When something goes wrong, they will analyze what happened and make changes so that it won’t happen again. They constantly think about how to make processes at work more efficient.

**In short, proactive people work in the present in order to improve things in the future.**

They are able to see the big picture, and are willing to put forth effort to improve it.

Sounds pretty good, right?

To learn the skill of being proactive, you’ll have to use some trial and error, and a lot of observation. But once you figure it out, the benefits will spill over into other areas of your life.

**In closing**

Thanks for reading these tips on how to be a good web developer! If you’re a newbie web developer, I really hope that you’ve found some of them helpful.

If you want a nice printable version of the list, you can pick one up [here](https://coder-coder.com/cu-uncommonly-good/).