8 Ways to Become a Better Coder

By Esther Schindler • Feb. 22nd, 2016 • Tech Topics

https://blog.newrelic.com/2016/02/22/8-ways-become-a-better-coder/

It's time to get serious about improving your programming skills. *Let's do it!*

That's an easy career improvement goal to give oneself, but "become a kick-ass programmer" is not a simple goal. For one thing, saying, "I want to get better" assumes that you recognize what "better" looks like. Plus, too many people aim for improvement without any sense of how to get there.

So let me share eight *actionable* guidelines that can act as a flowchart to improving your programming skills. These tidbits of wisdom are gathered from 35 years in the computer industry, many of which were spent as a lowly grasshopper at the feet of some of the people who defined and documented it.

1. Remind yourself how much you have to learn

The first step in learning something is recognizing that you don't know it. That sounds obvious, but experienced programmers remember how long it took to overcome this personal assumption. Too many computer science students graduate with an arrogant "I know best" bravado, a robust certainty that they know everything and the intense need to prove it to every new work colleague. In other words: Your "I know what I'm doing!" attitude can get in the way of learning anything new.

2. Stop trying to prove yourself right

To become great—not just good—you have to learn from experience. But be careful, experience can teach us to repeat poor behavior and to create bad habits. We've all encountered programmers with eight years of experience … the same year of experience, repeated eight times. To avoid that syndrome, look at everything you do and ask yourself, "How can I make this better?"

Novice software developers (and too many experienced ones) look at their code to admire its wonderfulness. They write tests to prove that their code works instead of trying to make it fail. Truly great programmers actively look for where they're wrong—because they know that eventually users will find the defects they missed.

3. “The code works” isn’t where you stop; it’s where you start

Yes, your first step is always to write quality software that fulfills the spec. Average programmers quit at that point and move on to the next thing.

But to stop once it’s “done” is like taking a snapshot and expecting it to be a work of art. Great programmers know that the first iteration is just the first iteration. It works— congratulations!— but you aren’t done. Now, *make it better*.

Part of that process is defining what “better” means. Is it valuable to make it faster? Easier to document? More reusable? More reliable? The answer varies with each application, but the process doesn’t.

4. Write it three times

Good programmers write software that works. Great ones write software that works exceedingly well. That rarely happens on the first try. The best software usually is written three times:

1. First, you write the software to prove to yourself (or a client) that the solution is possible. Others may not recognize that this is just a proof-of-concept, but you do.
2. The second time, you make it work.
3. The third time, you make it work *right*.

This level of work may not be obvious when you look at the work of the best developers. Everything they do seems so brilliant, but what you don’t see is that even rock-star developers probably threw out the first and second versions before showing their software to anyone else. Throwing away code and starting over can be a powerful way to include “make it better” into your personal workflow.

If nothing else, “Write it three times” teaches you how many ways there are to approach a problem. And it prevents you from getting stuck in a rut.

5. Read code. Read lots of code

You probably expected me to lead with this advice, and indeed it’s both the most common and the most valuable suggestion for improving programming skills. What is less evident are the *reasons* that reading others’ code is so important.

When you read others’ code, you see how someone else solved a programming problem. But don’t treat it as literature; think of it as a lesson and a challenge. To get better, ask yourself:

- *How would I have written that block of code?* What would you do differently, now that you've seen another solution?
- *What did I learn?* How can I apply that technique to code I wrote in the past? ("I'd never have thought to use recursive descent there…").
- *How would I improve this code?* And if it's an open source project where you are confident you have a better solution, *do it!*
- *Write code in the author's style*. Practicing this helps you get into the head of the person who wrote the software, which can improve your empathy.

Don't just idly think about these steps. Write out your answers, whether in a personal journal, a blog, in a code review process, or a community forum with other developers. Just as explaining a problem to a friend can help you sort out the solution, writing down and sharing your analysis can help you understand why you react to another person's code in a given way. It's all part of that introspection I mentioned earlier, helping you to dispassionately judge your own strengths and weaknesses.

Warning: It's easy to read a lot of code without becoming a great programmer, just as a wannabe writer can read great literature without improving her own prose. Plenty of developers look at open source or other software to "find an answer" and, most likely, to copy and paste code that appears to solve a similar problem. Doing that can actually make you a *worse* programmer, since you are blindly accepting others' wisdom without examining it. (Plus, it may be buggier than a summer picnic, but because you didn't take the time to understand it, you'll never recognize that you just imported a bug-factory.)

6. Write code, and not just as assignments

Working on personal programming projects has many advantages. For one, it gives you a way to learn tools and technologies that aren't available at your current job, but which make you more marketable for the next one. Whether you contribute to an open source project or take on pro-bono work for a local community organization, you'll gain tech skills and self-confidence. (Plus, your personal projects demonstrate to would-be employers that you're a self-starter who never stops learning.)

Another advantage of writing code for fun is that it forces you to figure things out on your own. You can't leave the hard stuff to someone else, so it keeps you from asking for help too soon.

Pro tip: Don't choose only personal projects where you never fail. You need to fail! But you do probably don't want to fail at work or when you have a deadline.

7. Work one-on-one with other developers any way you can

It helps to listen to other people. That might mean pair programming, or going to a hackathon, or joining a programming user group (such as the Vermont Coders Connection). When you contribute to an open source project, pay attention to the feedback you get from users and from other developers. What commonalities do you see in their criticism?

You might be lucky enough to find a personal mentor whom you can trust to guide you in everything from coding techniques to career decisions. Don't waste these opportunities.

8. Learn techniques, not tools

Programming languages, tools, and methodologies come and go. That's why it pays to get as much experience as you can with as many languages and frameworks as possible. Focus on the programming fundamentals, because the basics never change; pay more attention to architecture than to programming. If you feel certain that there's only one right way to do something, it's probably time for a reality check. Dogma can hamper your ability to learn new things, and make you slow to adapt to change.

I could keep going, but a key tenet of self-improvement is knowing when to stop.

Code, *learning*, *reading*, *and collaboration* *images courtesy of Shutterstock.com*.

careers, developers, programming, software development

Since 1992, Esther Schindler has made a living by translating from Geek into English. Find her on Twitter, Facebook, and Google+, where she's sure to distract you from getting productive work accomplished. View posts by Esther Schindler.