Dear KV,
I just started working for a new project lead who has an extremely annoying habit. Whenever I fix a bug and check in the fix to our code repo, she asks, “How do you know this is fixed?” or something like that, questioning every change I make to the system. It’s as if she doesn’t trust me to do my job. I always update our tests when I fix a bug, and that should be enough, don’t you think? What does she want, a formal proof of correctness?

I Know Because I Know

Dear I Know,
Working on software is more than just knowing in your gut that the code is correct. In actuality, no part of working on software should be based on gut feelings, because, after all, software is supposed to be a part of computer science, and science demands proof.

One of the problems I have with the current crop of bug-tracking systems—and trust me, this is only one of the problems I have with them—is that they don’t do a good job of tracking the work you’ve done to fix a bug. Most bug trackers have many states a bug can go through: new, open, analyzed, fixed, resolved, closed, etc., but that’s only part of the story of fixing a bug, or doing anything else with a program of any size.

A program is an expression of some sort of system that you, or a team, are implementing by writing it down as code. Because it’s a system, you have to have some way of reasoning about that system. Many people will now leap up and yell, “Type Systems!” “Proofs!” and other things about which most working programmers have no idea and which they are not likely ever to come into contact with. There is, however, a simpler way of approaching this problem that does not depend on a fancy or esoteric programming language: use the scientific method.

When you approach a problem, you ought to do it in a way that mirrors the scientific method. You probably have an idea of what the problem is. Write that down as your theory. A theory explains some observable facts about the system. Based on your theory, you develop one or more hypotheses about the problem. A hypothesis is a testable idea for solving the problem. The nice thing about a hypothesis is that it is either true or false, which works well with our Boolean programmer brains: either/or, black or white, true or false, no “fifty shades of grey.”

The key here is to write all of this down. When I was young I never wrote things down because I thought I could keep them all in my head. But that was nonsense; I couldn’t keep them all in my head, and I didn’t know the ones I’d forgotten until my boss at the time asked me a question I couldn’t answer. Few things suck as much as knowing that you’ve got a dumb look on your face in response to a question about something you’re working on.

Eventually I developed a system of note taking that allowed me to make this a bit easier. When I have a theory about a problem, I create a note titled THEORY, and write down my idea. Under this, I write up all my tests (which I call TEST because, like any good programmer, I don’t want to keep typing HYPOTHESIS). The note-taking system I currently use is Org mode in Emacs, which lets you create sequences that can be tied to hot keys, allowing you to change labels quickly. For bugs, I have labels called BUG, ANALYZED, PATCHED, I, and FIXED, while for hypotheses I have either PROVEN or DISPROVEN.

I always keep both the proven and disproven hypotheses. Why do I keep both? Because that way I know what I tried, and what worked and what failed. This proves to be invaluable when you have
a boss with OCD, or, as they like to be called, “detail oriented.” By keeping both your successes and failures, you can always go back, say in three months when the code breaks in a disturbingly similar way to the bug you closed, and look at what you tested last time. Maybe one of those hypotheses will prove to be useful, or maybe they’ll just remind you of the dumb things you tried, so you don’t waste time trying them again. Whatever the case, you should store them, backed up, in some version-controlled way. Mine are in my personal source-code repo. You have your own repo, right? Right?!

KV

LOVE IT, HATE IT? LET US KNOW
feedback@queue.acm.org

KODE VICIOUS, known to mere mortals as George V. Neville-Neil, works on networking and operating system code for fun and profit. He also teaches courses on various subjects related to programming. His areas of interest are code spelunking, operating systems, and rewriting your bad code (OK, maybe not that last one). He earned his bachelor’s degree in computer science at Northeastern University in Boston, Massachusetts, and is a member of ACM, the Usenix Association, and IEEE. He is an avid bicyclist and traveler who currently lives in New York City.