This is the Foo Field

The meaning of bits and avoiding upgrade bog downs

George Neville-Neil

Dear KV,

When will someone write documentation that tells you what the bits *mean* rather than what they *set*? I've been working to integrate a library into our system, and every time I try to figure out what it wants from my code, all it tells me is what a part of it is: "This is the foo field." The problem is that it doesn't tell me what happens when I set foo. It's as if I'm supposed to know that already.

Confoosed

Dear Confoosed,

Nowhere is this problem more prevalent than in hardware documentation. I am sure Dante listed a special ring of hell for people who document this way, telling you what something is while never explaining the why or how.

The problem with that approach is assumed knowledge. Most engineers, of both the hardware and/or software persuasion, seem to assume that the people they're writing documentation for—if they write documentation at all—already have the full context of the widget they're working on in their heads when they start to read the docs. The documentation in this case is a reference, but not a guide. If you already know what you need to know, then you're using a reference; if you don't know what you need to know, then you need a guide. Companies that care about their documentation will, at this point, hire a decent technical writer.

The job of a technical writer is to tease out of the engineer not only the *what* of a device or piece of software, but also the *why* and the *how*. It is a delicate job, because given the incredible malleability of software, one could go on for thousands of pages about the *what*, not to mention the *why* and *how*. The biggest problem is that the *what* is the easiest question to answer, because it is in the code when dealing with software, or the VHDL (VHSIC Hardware Description Language) when dealing with hardware. The *what* can be extracted without talking to another person, and who really wants to spend the day pulling engineers' teeth to get coherent explanations about how to use their systems? Since it is easiest to get at the *what*, most documentation concentrates on this part, often to the exclusion of the other two. Most tutorial documentation is short, and then at some point the rest of it is left as "an exercise to the reader." And exercise it is. Have you ever tried to lift a reference manual?

Although many engineers and engineering managers now give lip service to the need for "good documentation," they continue to churn out the same garbage that technical people have joked about since IBM intentionally left pages blank. A good writer knows that his or her job is to form in the mind of the reader a sense and an image of what the writer is trying to communicate. Alas, programmers and engineers have rarely been known as good writers; in fact, they are most often known as atrocious writers. It turns out that writers often want to relate, in some way, to people. That is, however, not something often said about technical folk, and in fact, it's often quite the

opposite. Most of us want to go off into a corner and “do cool stuff” and be left alone. Unfortunately, none of us works in a vacuum, and so we must at least learn to communicate effectively with others of our ilk, if only for the sake of our own project deadlines.

Every software and hardware developer should be able to answer the following questions about systems they are developing:

1. Why did you add this? (field, feature, API)
2. How is this field, feature, or API used? Give an example.
3. Which other fields, features, and APIs are affected by using the one you are describing.

And if the answer to #1 is, “Management told me to,” then it’s time to fire management, or find a new job.

KV

Dear KV,

During a recent rollout, I overheard one of our DevOps folks bemoaning the fact that upgrading our software had slowed down the overall system. This is a complaint I hear a lot, so I think it is happening more often. The problem is the folks at my gig don’t do enough performance testing and just upgrade systems whenever our vendors tell them to so that they won’t miss any new features, whether they use those features or not.

Bogged Down by Upgrades

Dear Bogged,

What you’re really seeing are 10-year-old expectations trashed by modern hardware trends. Everyone in computing has been talking about the end of frequency scaling for at least five years, and probably more. While lots of folks sounded the warning about this problem, and talked at length about the ways in which software would have to change to meet them, not enough software has been rewritten—oh, I’m sorry, I meant *refactored*—to handle this new reality. I am often amazed by people who upgrade software and expect it automatically to be faster.

Expecting more features makes some sense, because that’s what marketing and management are always going to push for in a new version of a system. The more boxes you can tick, the more money you can charge, even if the things provided are of little or no use. Given that upgrades always include new features, what makes anyone think that the system provided will run any faster? Surely more code to execute means that the system will run slower and not faster after the upgrade—unless you upgrade your hardware at the same time. None of which is to say that this *must* be the case—it’s simply that it often *is* the case.

The end of frequency scaling, the ever-upward tick of CPU frequencies, was supposed to spur the software industry into building applications that took advantage of multiple cores, as transistor density is still climbing, even if clock frequency is not. Newer software does seem to take advantage of multiple cores in a system, but even when it does, another problem is presented: memory locality. Anyone who has been building software on the latest hardware knows that the programs now need to know where they’re running in order to get fast access to memory. In multiprocessor systems, memory is now nonuniform, meaning that if my program runs on processor A but the operating system gives me memory nearer to processor B, then I am going to be very, very annoyed.

Modern operating systems are trying to handle NUMA (nonuniform memory access) correctly, but when they get it wrong, you become—as you signed this letter—bogged down again.

These are the new rules of the game programmers must contend with. Processors aren’t

getting faster; they're splitting into parallel machines with nonuniform memory. In the current environment, we now need to worry about all the things we may have last seen in parallel-programming classes in graduate school. All programming will now be threaded programming, and we'll have to deal with all that entails, plus the fact that we now need to know where our memory is coming from. My advice is to switch careers from programming to ditch digging (where at least at the end of the day you'll know you did something). If you can't switch careers, here are a few things you'll need to do and check as you try to improve the responsiveness of your code:

1. Learn to write correct threaded programs. Writing threaded code is hard, but there are plenty of books on this topic to help you.
2. Keep your threads from sharing state whenever possible.
3. Learn the APIs your operating system gives you to figure out where your thread is relative to the CPU and memory.
4. Bake debugging for threaded code into your system if it's not easily available as a library. There are few things more exquisitely painful to a software engineer than tracking down a race condition with `printf()` and a spoon.

Finally, find yourself a kind bartender who's willing to keep pouring long after he or she should—and remember to tip well.

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.

© 2013 ACM 1542-7730/13/1200 \$10.00