acmqueue

Standards Advice

Easing the pain of implementing standards

I would like to dedicate this column to my first editor, Mrs. B. Neville-Neil, who passed away after a sudden illness on December 9, 2009. She was 65 years old.

My mother took language, both written and spoken, very seriously. The last thing I wanted to hear upon showing her an essay I was writing for school was, “Bring me the red pen.” In those days I did not have a computer; all my assignments were written longhand or on a typewriter, so the red pen meant a total rewrite. She was a tough editor, but it was impossible to question the quality of her work or the passion that she brought to the writing process. All of the things Strunk and White have taught others throughout the years my mother taught me, on her own, with the benefit of only a high school education and a voracious appetite for reading.

It is, in large part, due to my mother’s influence that I am a writer today. It is also due to her influence that I review articles, books, and code—on paper, using a red pen. Her edits and her unswerving belief that I could always improve are, already, keenly missed.

George Vernon Neville-Neil III

Dear KV,

I’ve been implementing a network protocol for my employer, and although I’ve heard people complain about technical specifications before, the group that designed this one must be particularly special. Not only is the text nearly impenetrable, but I also keep finding that they have left out important points, such as whether or not some fields are supposed to exist in some cases. I feel as if I have no choice but to find another implementation and test mine against it so that I know if it will actually work, a step I was going to take later in the development process. How can anyone be expected to implement software based on such a document?

Duped by Documentation

Dear Duped,

You have a document?! Consider yourself lucky! Actually, perhaps you’re not really lucky. It turns out that the quality of standards varies just as widely as the quality of code, and a good way to find this out is to spend a couple of decades reading them, as KV has. If this is your first time implementing something from a “standard,” you probably expected it to be written by intelligent professionals who were focused solely on making sure that the people implementing their ideas were able to do so quickly and efficiently with the least amount of ambiguity. What you probably didn’t know was that such philosopher kings are the things of myth. Standards are written for many different reasons, some of them having nothing to do with the quality of the eventual product. There are even cases—and I know you will be shocked to hear this—where companies specifically

send people to work on standards so that they will either never see the light of day or when they do, they will be unimplementable, therefore giving that company a commercial edge. Of course, my telling you that people are stupid, vain, and vile and that companies are just as likely to destroy innovation as to promote it doesn't really help you, but it does feel good.

On a slightly more practical level, there are several things to note when implementing a standard in code. The first is that rather than go directly to interoperability testing, which you allude to, you should start by marking up the standard. Now that specs are usually issued in PDF, there are several good programs that allow you to keep arbitrary notes with the document. KV actually prefers the pen and paper method, although for some standards, carrying around a printed copy can be cumbersome. Whatever your markup tool of choice, go somewhere quiet, sit down, and read the entire spec and make notes. Call out every ambiguity you find. If your notes runneth over, keep a separate file of them somewhere. Think of this as being similar to writing comments in your code. Some standards and specs have commentary, and some of this commentary is even useful, but often these documents arrive more as pronouncements from on high, though perhaps with fewer *thous* and *thees* in them.

Once you have marked up the document, the next thing to do is write tests for all of the cases that you can tease out of the document. I know, I often go on about how important testing is, but in this case it's truer than any other I can think of. I personally had the displeasure of working with a networking standard in which the authors had not consistently declared their padding bytes. In some places they very dutifully said, "These bytes are always 0," but in others they said nothing. It was only after I had written several tests for this protocol that I determined that they had meant every one of their declared fields to start on a 32-bit boundary. Once I saw how the bytes would look on the wire, something else made nearly impossible by the horrible notation used in the standard, their original intent became more obvious. Most people call this an "Aha" moment, or, if they're in the bath, they yell, "Eureka!" I, and the people who sit near me at work, can tell you that "Eureka!" was not what I yelled. Let's just say that my dad was a sailor, and I curse like one. There is no way I would have been able to figure out this problem without my own test code.

We now come to one of the points you made in your letter: testing against other known implementations. If you're lucky enough to not be the first poor sap implementing the spec, then yes, interoperability testing may help you. I say *may* help because the person or group who implemented the code you're testing against may have been more confused than you are, so making your code work against theirs just means there are two interoperable, but also flawed, implementations of the same standard. Hurray for that. Do not assume that just because a version of the standard you're implementing exists, that it is any good. The world is littered with systems that are interoperable but that are also wrong. I am thinking here of the many cases of network clients that have to work with code from a large company in the Northwest. I don't normally go after a particular vendor in this column, but it has been my experience that one particular vendor has been the source of more crap networking code than any other I have come across, so enough said.

One last recommendation is that you specifically call out which section of a standard or spec is being implemented in the code. For example:

```
/*
 * Update the Older Version Querier Present timers for a link.
 * See Section 7.2.1 of RFC 3376.
 */
```

and

```
/*
 * RFC 1122, Sections 3.2.2.1 and 4.2.3.9.
 * Treat subcodes 2,3 as immediate RST
 */
```

Both of these examples come from the TCP/IP stack implemented in FreeBSD, but this is a common practice when implementing code directly from a standard or specification, and it helps to improve your life in a few ways. First, writing things down is one way in which people reason about problems. So long as things are only voices in your head, they don't have the same concreteness as they do on paper or in a file. Once something is out of your head and you can examine it more objectively, you can do a better job of reasoning about whether or not what you thought was actually the case.

Second, these act, as do all good comments, as sign posts to people who will maintain the code. There is nothing more frustrating than looking at some obscure piece of a function and wondering, "Now why did they do that?"—particularly if what was done doesn't make immediate sense. The code probably exists for a good reason, but it's important to separate the original programmer's capriciousness from the capriciousness of the standard. If it's a part of the standard, then it might look irrational, but for the sake of interoperability you'll have to leave it alone.

Of course, as you can imagine, I have some advice for standards writers as well. Perhaps the most important thing anyone working on a standard or spec can do is to be consistent in language and representation. At this point most of the constructs that would go into a new standard already exist, so please, stop inventing new ways to represent data structures. I'm sorry if you find it exciting to come up with new ways to represent bytes and bits on paper, but standards are not works of visual art; they need to be works of clarity. I happen to prefer the textual representation found in most RFCs, where you get labeled boxes, at most 32 bits wide. These are not the be-all and end-all of visual representation, but they're a damned good start, so please, start there.

Now, once you think you have a clearly written document, hand it to someone who is not working on your project and see if it really is clear. The idea that a group of people working closely on a standard are the right people to check the standard for clarity is ludicrous. Even after only cursory exposure to the set of ideas in the standard, an internal reviewer's brain will begin to fill in the blanks, and that's absolutely not what you want. You want someone who will call out the blanks and tell you where they are. Finally, have someone try to implement the spec—again, someone external to the group writing it—and then LISTEN TO WHAT THEY SAY! Far too many times I've asked someone, "Did anyone review this?" and they said, "Yes, of course!" in a shocked tone as if I'd

asked if they had showered that day and was impugning their sense of hygiene. And then when I asked, “Did you integrate their feedback?” they began to look quite sheepish.

Implementing a standard isn’t too different from any other sort of implementation. The authors of something marked as a standard are not gods, and their utterances should not be taken as commandments. The short answer to your question is take notes, write tests, and keep a bottle of your favorite sedative nearby, because if you don’t need it this time, well, you will eventually.

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.

© 2009 ACM 1542-7730/09/1200 \$10.00