acmqueue

Code Rototilling

KV hates unnecessary work

Dear KV,

Whenever a certain programmer I work with needs to add a variable to a function and the name collides with a previously used name, he changes all of the previous instances to a new different name so that he can reuse the name himself. This causes his diffs to be far larger than they need to be and annoys the hell out of me. Whenever I challenge him on this, he says that the old usage was wrong, anyway, but I think that's just him making an excuse.

Vexed by Variables

Dear Vexed,

This kind of behavior is just another childish version of the not-invented-here syndrome or, as Nirvana called it, "Territorial Pissings." You're quite right to challenge this sort of antisocial behavior, because it leads to that bane of bug tracking: unnecessary diffs.

One of the goals of good source-code maintenance is to minimize unnecessary textual changes so that the important part of a change is easy to find. Although it might be a nice idea to spruce up a bit of code one finds to be ugly, unless the code in question is frequently the source of problems, it's usually not worth the effort. If it works, don't fix it. If it doesn't work, well, you'll probably have to rip out the whole function, module, or subsystem and start again.

It does amaze me that there really is very little middle ground between bug fixing and code rototilling, but I've found that once I've fixed the 10th bug in the same 100-line section of code, my temptation "just to fix what's broken" breaks down, and I know that it's time to rip off the Band-Aids and do some surgery.

The only other reason programmers make textual changes that don't really relate to functionality is so they can pretend to be doing work. On open source projects this is referred to as the "token commit"—making a minor change just so that your name winds up in the project history. In a corporate environment, it's a way of boosting your KLoc—the number of lines you've changed since your last review. Either way, it's annoying and antisocial, and I support your effort to stomp it out. Emphasis on *stomp*.

KV

Dear KV,

One of the people I work with started a new project that depends on a large open source project. In order to track the external project, my colleague stored it in our internal source-code control system, and now every time anyone tries to update the code, the client has to traverse a massive project just to make sure all the other files are up to date.

Version Indigestion

Dear Version,

Unless there are local changes to the external project, there is no reason at all to store it in your version-control system. That being said, it's quite common—and even encouraged—for projects to depend on other projects. Most software is built by standing on the shoulders of giants, and clearly your project picked a tall one to stand on.

The right way to go about this, of course, is to give each external project its own local repository. You can explain this to your coworker as a form of modularization. Each external component is seen as a module for the system and is maintained in its own separate area. Splitting out each project makes it far easier to update the project when the inevitable next release comes along. Being able to preserve the external project's commit history without it becoming intermixed with yours is just an extra side benefit of following this procedure.

Another reason not to commit someone else's source code to your internal repository is that—with the proliferation of source-code control systems—the other system may not be the same as yours, and it may not even follow the same commit model. Some people use centralized systems with monotonically increasing commit numbers, such as subversion, while others use distributed systems with hashes such as mercurial and git. Attempting to map commits from one system to another is possible, as git-svn shows, but in this particular case, it is unnecessary work. KV hates unnecessary work.

One final and often-overlooked reason not to mix code bases is a legalistic one. Some code is licensed in such a way that modifications must be fed back to the original authors. Although mixing code in a single repository is not referred to in these licenses, the fact is that a programmer looking at a single repository will treat the code contained in it as “all one thing,” and when the programmer does that, he or she is bound to accidentally mix in your—possibly secret—sauce with the imported code. Good fences make good neighbors in source code, as well as in real life.

Clearly it's time to kick this code out of your repository and maintain it somewhere else.

KV

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.

© 2011 ACM 1542-7730/11/1100 \$10.00