acmqueue

Lazarus Code

No one expects the Spanish Acquisition

Dear KV,

I've been asked to look into the possibility of taking a 15-year-old piece of open-source software and updating it to work on a current system used by my company. The code itself doesn't seem to be too bad, at least no worse than the code I'm used to reading, but I suspect it might be easier to write a new version from scratch than to try to understand code that I didn't write and which no one has actively maintained for several years. What is the point at which I should decide to ignore this old code and write something new?

Lazarus

Dear Lazarus,

Ah yes, GitHub, or was it SourceForge, where old code goes to die, but never quite does. There are probably more abandoned software projects online now than there are projects that are actually used. One of the problems with access to cheap storage is that software engineers, many of whom are not so secretly also data hoarders, need never delete anything. History now will never be forgotten, more's the pity.

Whether or not to resurrect a piece of software depends on a number of variables, including how soon you need the code, how well the code itself worked in the past, and how easy it will be to mentally ingest what you aim to use. I won't address the time question because we know that everyone needs whatever it is yesterday and arguing with management that writing new code will be faster than reusing extant code is a losing battle, one that leads to meetings where you state your name and the drug that you've been abusing ever since you started whatever project it was.

From a technical standpoint, you have a few good tools that can tell you how hard it will be to resurrect a piece of code, and the first is your compiler. Try to build the code with all the warnings and errors set to their strictest levels. If the code builds without problems under these strictures, then one of two things is true. First, the code may have been written by someone who really knew what they were doing and who was quite careful in making sure their code not only adhered to the current language standard, but also that it used a narrow enough subset of the language that changes in the language did not cause any problems. The second possibility is that your compiler is broken. It's actually quite rare that a 15-year-old piece of code will compile completely without warnings under a newer compiler. The number of warnings and errors generated will give you a first indication of how much work you face.

Does the system have a set of tests that were written for it and checked into the repository? If so, you are in luck, and you should go out immediately and play your local lottery. A lot has been written about testing over the last several decades of software development, some of it by KV, and, from time to time, someone actually builds and maintains a good set of unit tests for their software. If these exist, you should read them first, because test code often gives a much deeper insight to what the system is supposed to do than the comments in the actual code.

You say that the code is stored in an online repository. If you've gotten past the first two

suggestions above then you should read the commit logs for the files that contain the main data structures. If you can't find a simple set of files that contain the data structures, then it's probably time to start again. Good commit logs tell a lot about what a programmer is thinking when developing code. Since data structures are the central place from which all the rest of the code should flow and to which it should always refer back, the logs for these files are of paramount importance. An example of an acceptable message is shown below:

Added the ability to store data in offline files. Included both a file name and a file descriptor in the header data structure. The filename is auto-generated by the `tmpfile()` API if it is not supplied as an argument or in a configuration file by the user.

Commit logs that contain short lines such as "fixed bug" are a warning to stay away. It's fine to see that very rarely, but I expect something a lot more descriptive, such as a bug number or link to a bug tracking database as well as a complete description of what the problem was and how the fix addressed it.

Many older pieces of software depend on older versions of operating system and library APIs. That is not a reason to rewrite the code, as many of those changes are mechanical in nature. Deciding how to address the changes in library APIs is a judgment call on the part of the programmer, but those changes should never be grounds to abandon a piece of software unless the library or API itself simply no longer exists.

I would hope this goes without saying, but I've gotten so used to having to say such things anyway that I'll say this here. Before you make any changes to the system, make sure to make a copy of the repo and to also import the repo into a fresh repository before you start working on it. Use a tag or similar construct in your source code control system to indicate where the old code stopped and where any of your changes might begin. Otherwise you will be in the unenviable position of not being able to explain where the code came from to the next person who has to maintain it.

KV

Dear KV,

I've been working for a medium-sized software company for a few years and have been recently "promoted" to software architect. This seems to mean that I do all the coding I used to do, but also have to explain to everyone else on my team what to code. Last month I was asked by management to look at the software from a potential company acquisition and to advise them on whether we should or should not acquire the company based on the software that they have already written. This seems like a straightforward case of code review, but is it?

Acquisitive

Dear Acquisitive,

The first thing to do is to become close friends with the engineers at the company, because your company is about to make them rich and then they're all going to quit and you're going to have to maintain their code, without the benefit of a bonus or stock payout. Also, in my experience, no matter what you say, management will do whatever they want, but as an engineer you have to do your best. So let me see if I can come up with a more useful answer.

You are correct that what you're being asked to do is effectively a code review, but it's one with quite a few strings attached and one that is also being done under very different circumstances than a review that might be done within your company. Think of this code review more as a legal deposition. Legal depositions, I am told, are not like those fun courtroom dramas, where two lawyers ask polite questions trying to trick the person under examination into giving away information that shows their guilt. It is more like an inquisition where multiple lawyers get to ask any question they like and in any tone they prefer. In this case, you should act like one of those lawyers. Your job is to get as much information out of the engineers at the target of the acquisition as necessary, because, in three months, you'll be maintaining their code. Don't let them fob you off with a big zip file of code. Make sure you get not only code, but access to their tools, repository, and engineer time. Make one or more of them sit in a room with you for as long as necessary to walk through what the system is, how it's supposed to work, and where that happens in the code.

Also make sure to arrange several after-work meals with the engineering group, and get approval to buy as much alcohol as you think you'll need to get them to tell you where the real bodies are buried in the code. That last suggestion is not meant in jest, it is deadly serious. If you want to get information out of engineers, feed them and get them drunk. Bring along at least one other engineer from your company, one who you trust and who can take notes and also play "good cop." When you ask a hard question, your good cop is supposed to soften whatever you said so that the team you're interviewing actually becomes willing to comply with your requests. I've considered hiring someone to just follow me around playing "good cop" to keep me out of trouble, but I think at this point in my career an entourage is a tad premature.

Finally, do not believe in demos. Demos can and will be faked. Make sure you can poke at the system and code before making any serious recommendation to management. Once you've made the recommendation, be prepared to have to work with the code, because even if you say that the system contains a function that when executed will spell the end of all humankind, that may not stop management from deciding that the deal is, indeed, a good one, and then stick you with the problem of making it work.

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.

© 2015 ACM 1542-7730/15/0400 \$10.00