Removing Kode

DEAD FUNCTIONS AND DEAD FEATURES

Dear KV,

Your columns on koding often talk about issues that come up when *adding* new code, but when is the right time to *remove* a piece of code? I've been working on an enormous legacy code base and most of the questions that come up in our stand-up meetings concern which modules we think we can remove from the system.

Ready for Removal

Dear Ready,

I'm surprised and amused at the narrowness of your question. Most koders, when confronted with a legacy code base, simply want to throw it out and start again. I have to say that this happens to me at least once a week, just looking around at the astounding mountains of legacy that pass as software.

Removing dead code from systems is one of KV's favorite koding pastimes because there is nothing quite like that feeling you get when you get rid of something you know wasn't being used. Code removal is like cleaning house, only sometimes you clean house with a flame thrower, which, honestly, is very satisfying. Since you're using a version-control system (you had better be using a VCS!), it's very easy to remove code without worry. If you ever need the code you removed, you can retrieve it from the VCS at will.

Complications arise when you have code that is infrequently used but has been in the system for a long time.

As to when you should be removing code, there are several answers. The first, of course, is that if you have truly dead code that can no longer be reached from anywhere in the system and that is not simply conditionally compiled-out test code (see my 2016 column, “The Unholy Trinity of Software Development,” <https://queue.acm.org/detail.cfm?id=3014148>), then that code should be removed immediately. A good compiler or other tool will tell you when you have dead code, which should make the job fairly straightforward.

Complications arise when you have code that is infrequently used but has been in the system for a long time. At this point, you have to do a bit more thinking about your code and whether or not the code you’re looking at is—for want of a better term—*nearly dead*. Any system that survives for a number of years will tend to grow functionality that is used—sometimes briefly—and then discarded or ignored. The logic for not removing dead features is usually stated as, “If it ain’t broke...,” which is actually quite foolish. If your software is so fragile that removing a feature breaks the whole system, you have much bigger problems than needing a bit of code cleanup, and that means the system as a whole probably needs rototilling if not an outright rewrite.

A dead *feature* is different from a dead *function*. Dead features are either completely unused or used by only a minority of users. The risk with dead or nearly dead features is that they leave a larger attack surface in your code. Once upon a time, we also cared that dead features left code in the executable that bloated the system,

but only those of us working in the embedded-systems area seem still to care about that. For modern server-based systems, it is the risk of the dead feature giving an attacker a place to infiltrate your system that is probably preeminent in your mind, or at least it should be.

Sometimes a dead feature will have a noisy minority of users or an internal champion who has tended that feature for many years. At this point, the removal becomes a political (i.e., human) problem.

Related articles

Kode Vicious

Code Hoarding

Committing to commits, and the beauty of summarizing graphs

<https://queue.acm.org/detail.cfm?id=2897034>

Surviving Software Dependencies

Software reuse is finally here

but comes with risks.

Russ Cox

<https://queue.acm.org/detail.cfm?id=3344149>

Kode Vicious

Writing a Test Plan

Establish your hypotheses, methodologies, and expected results.

<https://queue.acm.org/detail.cfm?id=3294770>

There are several ways to solve political problems at work, but some of those will get you 10 to 20 years in jail if you carry them out. If the noisy minority is very small, it's possible that feature could be broken out into its own separate program so that the code is no longer part of the larger whole. If the problem is developers, well, it's time to give them a new challenge, far away from their pet features. Once you have developers with pet features, you have a very different sort of management problem, one that we will not address here.

You'll notice I didn't give you a definite timeline in answer to your question, and that's because there really isn't one other than removing

truly dead code as soon as it's dead. For feature removal, that really depends on how you deal with your users and developers. A good rule of thumb, though, is to remove features only at a major release so as to limit the level of surprise.

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. Neville-Neil is the co-author with Marshall Kirk McKusick and Robert N. M. Watson of *The Design and Implementation of the FreeBSD Operating System* (second edition). He is an avid bicyclist and traveler who currently lives in New York City.

Copyright © 2020 held by owner/author. Publication rights licensed to ACM.