Kode Vicious vs. Mothra!

In this month's installment, Kode Vicious tackles everything from `goto`-mongering to language snobbery. All in a day's work, as they say. If he weren't so vehemently opposed to it, we'd put him in tights and a cape with a big "KV" on his chest and send him out to fight Mothra, where he'd personally see to it that all of your koding konfrontations were put to rest once and for all. On the other hand, it might be safer for all involved if he sticks to his e-mail account, which, by the way, is kv@acmqueue.com.

Dear KV,

My co-workers keep doing really bad things in the code, such as writing C++ code with macros that have `goto`'s that jump out of them, and using `assert` in lower-level functions as an error-handling facility. I keep trying to get them to stop doing these things, but the standard response I get is, "Yeah, it's not pretty, but it works." How can I get them to start asking, "Is there a better way to do this?" They listen to my arguments but don't seem convinced. In some cases they even insist they are following good practices.

Help

Dear Help,

The scariest thing in your letter is the use of `goto` to jump out of a macro. I'm not even sure how your co-workers are using `goto`, as it sounds totally bizarre to me. Macros are supposed to be local bits of code expanded by the compiler at build time, and their side effects should be local. Much has been written about the use of `goto` since it first appeared on the scene, so I won't go back over that ground. I'm also not a total anti-`goto` fascist: I have seen code and worked with code that uses `goto` effectively. Using `goto` and a label to perform error cleanup in C is

Got a question for Kode Vicious? E-mail him at kv@acmqueue.com—if you dare! And if your letter appears in print, he may even send you a Queue coffee mug, if he's in the mood. And oh yeah, we edit letters for content, style, and for your own good!

A koder with attitude, KV ANSWERS

YOUR QUESTIONS.

MISS MANNERS HE AIN'T.

a long-held tradition in many places and can be used cleanly. Other uses are highly suspect, and

I would expect your co-

workers to justify them with references to well-written documents, such as papers or standards. Just saying, "This is a good practice" doesn't make it so. You need to be prepared to call their bluff when they use this old Jedi mind trick: "This is not the code you're looking for."

One of the most endearing and also annoying features of most koders I know is an overwhelming sense of self-confidence. My grandmother used to say, "Stubborn people are stupid people," and I've come to believe her. The problem you face is not just technical but also social. The question I most often ask myself in these situations is, "Where can I hide the bodies and when will it be most appropriate to leave town?" No, wait, that's not it.

Convincing people that they're wrong and you're right is very difficult and not always that much fun, unless you get to write about it for a magazine. I suggest patience on your part, which is OK for me because I don't have to take my own advice.

Other suggestions are to ask questions instead of giving answers. Questions such as, "Are you an idiot?", while good for the soul, are not good for the work environment. What I mean is to ask someone: "How do you debug code like that?" or "I'm really interested in that design pattern; where did you learn it?" If you can get inside their minds, tight squeeze that that might be, then you're going to be in a better position to change them.

KV

Dear KV,

I have a simple question for you: What is the correct handling of an error return from the `close()` system call?

Just Wondering

Dear JW,

I gather that your question is referring to the letter I received from Pointless Returns in the December/January 2004-2005 issue of *Queue*, in which I extolled the virtues of checking all return codes. Although you might

think that there is nothing to be checked when closing a file descriptor, that's just not true. File descriptors are limited per process on most operating systems that I work on, and I suspect on those you work on as well. Properly returning this resource to the operating system is as important as freeing memory that has been allocated.

Looking at the `close()` system call on Mac OS 10.3, the operating system that I use to write these little missives, I see two possible errors that it can tell you about. The first is that the file descriptor you tried to close wasn't really active. Well, that certainly is something to notice. It means that your program messed up somewhere and this is something that ought to be logged by the application or possibly cause the application to exit with an error.

The second possible error is that the `close` call was interrupted and did not, therefore, succeed. In a long-running program, the loss of file descriptors through `close` failing to complete its work would cause the program to crash at some future point, and to crash in a way that is hard to diagnose. It might take days to run out of file descriptors, and without checking the return code from `close` in every place that it's used, you would never know why.

So, I stand by the advice I gave to Pointless Returns. It's important to check the return codes from any function or system call that provides them.

KV

Dear KV,

When I read your column, you sound to me like one of these guys who *kodes* (as you would misspell it) only in C or maybe in C++. Many of us *kode* in other languages, such as PHP, Python, and Perl. How about writing about those languages—perhaps something written and designed after most of your readers were born?

Where I work we provide a lot of Web services so we use a good deal of PHP, with only a small amount of C and C++ doing the computationally intensive tasks, or things that have to be closer to the operating system. Do you have any advice for those of us who *kode* in other languages?

21st Century Koder

Dear 21,

I'll have you know that I was born after C, but only just. I have no doubt that many of the readers of *Queue* cut their teeth on other languages—perhaps, though I shudder to think, it was Basic—but I can write about only what people ask about. There have been letters from *kodors* using languages other than C or C++, such as Linguistically Lost, who wrote to me in the November 2004 issue.

If you had asked a concrete question, this would have made my job easier, but clearly that was not your goal. As for my misspellings, take it up with my editor, though I recommend you take a couple of self-defense classes first.

I have read plenty of PHP code in my time, as well as Python, Perl, C, C++, TCL, Fortran, Lisp, Cobol, and others. The basic fact is that what separates good code from bad code has very little to do with the language itself. As Donn Seeley recently pointed out in *Queue*, you should learn "How Not to Write Fortran in Any Language" (December/January 2004-2005).

Good code uses the dominant metaphors in a language to make it easy for other people to understand. All the languages I've mentioned here have comments, yet many people either leave these out or misuse them completely. Since the 1980s it has been possible to write understandable variable and function names, yet people continue to use single letters, believing that those who look at the code after them will know what they mean.

In PHP you can write this:

```
function getn($data)
as easily as this:
// This function takes a string as input in the name_field
// The name must be a string starting with an alphabetic
// character, i.e. A..Z or a..z, and may not be more than 32
// characters in length. Only the first character must be
// alphabetic and all the following characters can be
// alphabetic or numeric, i.e. 0..9
function get_name($name_field)
```

Nonetheless, people continue to write the first version. So, I don't care how modern you and your young friends are—you must apply the basics of writing code that is easy to understand the first time some other poor *koder* has to read it. Once you're done with that, get back to me with some specific questions on PHP that will illuminate its sleek, modern feel, and then we'll have something to talk about.

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 has made San Francisco his home since 1990.

© 2005 ACM 1542-7730/05/0500 \$5.00