Kode Vicious to the Rescue

Koding problems driving you nuts, ko-workers making you crazy? Never fear, Kode Vicious is here—to answer your questions, solve your problems, and just basically make the world a better place.

Dear Kode Vicious,

Where I work we use a mixture of C++ code, Python, and shell scripts in our product. I always have a hard time trying to figure out when it's appropriate to use which for a certain job. Do you code in only assembler and C, or is this a problem for you as well?

Linguistically Lost

LL,

First of all, I may be a grotty old koder, but I'm not so limited as to program in only C and/or assembler. If you ever suggest that again, I'll beat you senseless with this 6502 manual.

Now, choosing a language to use is not an easy task. The majority of coders just use what they're given at work and don't question that. Not questioning is bad. We should always try to find the right tool for the job. Then, once we've found the right tool, we can spend endless hours yelling at our co-workers for not using the same tool we do. No, wait, that's not it.

Let's start from the end of your list. Shell scripts, though sometimes clever and useful, are very difficult to maintain. Uncle Vicious never uses them for anything more than 100 lines of code. Why? I've seen products built on directories filled with shell scripts, and they invariably turn out to be a tangled mess. I'm sure there are people out there who have the most beautifully built, finely tuned shell scripts, and they will all send me nasty hate mail. Which, thankfully, will be electronic and easily deleted. The fact is that the majority of people who write shell scripts write them as one-off (or two-off because

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 t-shirt, 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.

they never work the first time) things to “just get something to work.”

Unfortunately these scripts

are the neglected children of projects. They are checked in and ignored until they cause a problem. Then they're beaten soundly until the screams die down and they begin to limp along again.

```
while I do
echo "previous" rant
done
```

Python, one of the modern interpreted languages, along with Perl, Ruby, and others, is an appropriate tool with which to replace shell scripts. These, too, have their limitations, most of which have to do with performance. Please, do not use Python to write device drivers, embedded systems, flight control software, or other code that gets between me and the brakes of my car. I tend to prototype things in Python, and then if I decide I need the speed, I move on to a compiled language. I'm sure that several people will write in to say that I should use Java, but I find that Java requires too much other craft (libraries, IDEs, etc.) for me to use it in the way that I use Python. Someday I might write a large system in Java, just like someday I might write a large system in Modula-3, Smalltalk, or Scheme. This is not what I use Python for. Python is for replacing hundreds of twisty shell scripts, all alike.

Finally, we come to C++. Now, whatever people may say about C++ with respect to other compiled languages, the fact is that it is used by a majority of koders in their day jobs. I prefer C++ for large, complex systems that are pretty static once they're in place. The fact is that when you're at work it is unlikely that you'll be able to say, “I don't think we should use C++ for this project,” unless you're the CTO—and given your question, I rather doubt you're the CTO.

I think one of the things you're really asking is when to put something into C++ vs. Python. If you have to write a class for general use in this system and others, then you probably want to code it in C++. Furthermore, if the code in question should not be modified or seen by the customer, then C++ is, again, your natural choice. I'm pretty sure your boss doesn't want you sending code snip-

pets of the company's secret sauce out with the product in easy-to-read chunks. The performance question is also important. Is the cost of starting up an interpreter and then interpreting the code going to outweigh the benefits of writing in an interpreted language? If so, then you're going to have to write that code in C++.

Brian W. Kernighan and Rob Pike actually did a good job of addressing issues like this in their book, *The Practice of Programming* (Addison-Wesley, 1999), which is a KV must-read.

So, there are no hard and fast rules here, of course, but this is how I handle these issues in my own work.

KV

Dear Kode Vicious,

We're starting a new project at work and are trying to decide on a coding standard. Of course, everyone on the team (there are 10 of us) wants to use their own personal favorite, or the one they used in their last company. How do we choose one and does it really matter?

So Many Standards

SMS,

Let me ask you: just how much time has your team spent deciding on a coding standard? A day, a week, a month? Don't you people have anything better to do?! I mean, come on, we've all been over this time and again and no one likes the answer, but an answer needs to be had. Get together, get some beer, have everyone bring along their favorite coding standard. Then lock the door and no one leaves until you've agreed. Luckily, the beer will make the decision go quicker, because, well, if you lock the door, someone is going to have to go, you know? Well, perhaps that's not the most productive way but at least it gets you a reasonably quick result.

The point of a coding standard is so that all the poor bastards you work with will be able to understand, hopefully, what your sleep-deprived minds created at 2 a.m. some Sunday morning—you remember, the code you created the day before a release. This understanding will become necessary at a later date. Perhaps when that first bug report comes in, about a day after the release, at 3 a.m. while you're catching up on the sleep you missed, or better yet, drinking down your most recent paycheck. So just remember, the poor bastard you help could be yourself.

If I were you, I'd go for readability, readability, and readability.

A good coding standard:

- Makes correlated chunks of code look like they're related, and not like the offspring of close relatives.

- Uses enough white space so that those of us who wear glasses (and you kids out there, you'll be wearing them too, soon enough, so stop laughing now!) can tell which code block goes with which if, while, or case statement. It should also not use so much white space that you need a new 23-inch monitor just to read the code.

- DoesNotMakeItSoThatNamesAreSoLongTheyLookLike-Sentences. This practice, in particular, makes me do things that I can't mention here, and that cost a lot of money if you have to pay for them. The idea, like so many ideas, has a kernel of goodness—that the reader of the code should be able to read what it does like written English text. But in the end you can't tell

```
ReadInputFileFromDiskIOComplete
from
```

```
ReadInputFileFromNetIOComplete
```

as well as you'd like. Picking rational names is more important than reproducing William Burroughs in a device driver.

- Makes it clear when there is a coding error. If the way the code is written hides things, like accidentally open else statements (you know, you thought you added a line of code to the else block, but forgot the braces?), then it's not helping you.

I say you start from there. The coding standard itself should be only four pages long. Most engineers don't want to read a loose-leaf notebook full of process BS to write code. Each statement in the coding standard must be written so that it is unambiguous (yes, it's a big word, look it up). Think of it as a list of do's and don'ts. Just say how routines are named, how code is indented, where braces are located, and what variables should look like—and be done with it. Then get back to work and stop drinking my beer!

Oh, and one more thing, I've always laughed at the comment, "What if you get hit by a bus?" My co-workers rarely get hit—by buses.

KV

Kode Vicious, aka 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 the ACM, the Usenix Association, and the IEEE. He is an avid bicyclist and traveler who splits his time between Tokyo and San Francisco.

© 2004 ACM 1542-7730/04/1000 \$5.00