The Obscene Coupling Known as Spaghetti Code

**TEACH
YOUR JUNIOR
PROGRAMMERS
HOW TO
READ CODE**

who is
KV?

click for video

Dear KV,

Forgive me, for my ACM membership has lapsed, and for my sins I have been saddled with mentoring a spaghetti coder.

I'm working on a piece of new software, green-field for once, but with stiff reliability requirements. My helper, a young, self-proclaimed "devop," aims to improve as a programmer, and, unfortunately, this person got stuck with me.

No matter how hard I constrain the work I dole out, I just can't stop this helper from the obscene coupling known as spaghetti code, all masquerading under obsessive, perfect syntax. We can't even get into the hard reliability aspects of the software, because tangled messes that lint perfectly and break opaquely just keep piling up.

After many approaches, each one narrower in scope than the last, I've come down to doling out work units that are constrained to writing single, well-defined functions in a Python library, but even then I'm failing to keep this person from needlessly chaining functions, silently mixing and transparently passing data through multiple layers of interfaces, and, most painfully, burying important error output in ways we all know too well as spaghetti code.

Assuming this apprentice is willing and eager, how can one go about breaking this fundamental coupling mentality in implementation and open this person's mind to engage the actual problem at hand—what the software does!

I don't want to botch this and produce the next

Darth Vader!

Mr. Function Defines Form

Dear Function,

Well, at least you didn't mention goto, the root of much of the spaghetti code of my well-spent youth. Yes, KV was once young, but because of programmers such as your ward, he has never looked young or beautiful.

Once upon a time, spaghetti code was defined by the fact that it jumped all over the place without any rhyme or reason, but, as you say, you have someone who, even when given a constrained contract such as single functions, is still able to make a plate of pasta of it.

Perhaps it's time to introduce the idea of *narrative* to your Padawan. Code, as I've pointed out countless times, is a form of communication between the people who write and maintain it and is only incidentally executable on a machine, which we call a computer. I cannot seem to say this often enough, clearly, because I say it a lot. Someday I'll lose my voice, and the people I'm screaming at will finally think they'll get some peace; but if that ever happens, I have a recorded version I can play through a megaphone.

Communication is just a fancy word for storytelling, something that humans have probably been doing since before we acquired language. Unless you are an accomplished surrealist, you tell a story by starting at the beginning, then over the course of time expose the reader to more of the details, finally arriving at the end where, hopefully, the reader experiences a satisfying bit of closure. The goal of the writer (or coder) is to form in the mind of the reader the same image the writer had. That is the process of communication, and it doesn't matter if

The most important part of any of these programs is that they do one thing, they do it clearly, and it is obvious to even the most inexperienced programmer what is going on.

it's prose, program, or poetry—at the end of the day, if the recipient of our message has no clue what we meant, then all was for naught.

Of course, as many brilliant writers have proven over time, clear narrative isn't entirely necessary, but let's just stick with the clear narrative metaphor for code, rather than claiming we should write an accounting system based on *Naked Lunch*. I mean, I'd enjoy it, but would it work? Only the Mugwumps would know.

The concept of simple narrative can be applied to code in the following way. We are trying to write down the steps that are required to do a particular job with a machine in such a way that when other readers come upon the narrative (code)—which is usually thrust upon them with a bug list as long as a baby's arm—they are able to pick up the story wherever they choose. For a short program, something less than 100 lines, the narrative can all be in one `main()` function. I recommend that you find a few such programs—well written, well commented, and that do one thing and do it well. Then make your Padawan read them and explain them to you.

KV has extolled the virtues of reading good code as a way of learning to write good code, and for young readers, short programs are best. Even though you are working on green-field code, a rarity in our industry, there must be some scripts or code lying about that you don't hate and that extol the virtues you wish to instill in this apprentice. The most important part of any of these programs is that they do one thing, they do it clearly, and it is obvious to even the most inexperienced programmer what is going on. Find that code, explain its beauty, and then make them extend and maintain it.

Since you both are working on the same code base, you also have ample opportunity for leadership by showing this person how you code. You must do this carefully or the junior programmer will think you're pulling rank, but, with a bit of gentle show and tell, you can get your Padawan to see what you're driving at. This human interaction is often difficult for those of us who prefer to spend our days with seemingly

logical machines. Mentorship is the ultimate test of leadership and compassion, and I really hope you don't wind up sliced in half on the deck of a planet-smashing space station.

KV

Related articles

Human-KV Interaction

Kode Vicious

A koder with attitude, KV answers your questions. Miss Manners he ain't.

<https://queue.acm.org/detail.cfm?id=1122682>

Reading, Writing, and Code

Diomidis Spinellis

The key to writing readable code is developing good coding style.

<https://queue.acm.org/detail.cfm?id=957782>

A Conversation with Steve Bourne,

Eric Allman, and Bryan Cantrill

Three Queue editorial board members discuss the practice of software engineering.

<https://queue.acm.org/detail.cfm?id=1413258>

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.