Annette Vee: Coding Values

Annette Vee, University of Pittsburgh

Today I want to talk about good code. Experienced programmers often think about what good code is. But they rarely agree.

And here’s what I want to say: they don’t agree on what good code is because there is no good code. Or, rather, there is no Platonic Ideal of Good Code. Like writing, there is no good code without context.

Unfortunately, when good code is talked about, it is often talked about as if there’s no rhetorical dimension to code. It’s talked about as though the context of software engineering were the only context in which anyone could ever write code. As if digital humanists, biologists, web hackers, and sociologists couldn’t possibly bring their own values to code.

I’ll give you just a couple of examples of how this happens, and what this means for us in computers and writing.

One of the earlier articulations of the supposed Platonic Ideal of Good Code was Edsger Dijkstra’s infamous “GOTO considered harmful” dictum, from 1968.

This article railed against unstructured programming and the GOTO command for its ability to jump from one place in a program to another without logical program flow. Dijkstra’s assertion was so provocative that "considered harmful" soapboxes have proliferated in programming literature and the Web and the “harm” done by goto has been hilariously depicted in an XKCD comic. Many of us first learned the joy of coding through the
languages that used the GOTO command such as BASIC. But Dijkstra’s statement suggests that the context of the software production workplace should override all other possible values for code. This is fine—as far as it goes, which is software engineering and computer science. But this kind of statement of values is often taken outside of those contexts and applied in other places where code operates. When that happens, the values of hacking for fun or for other fields are devalued in favor of the best practices of software engineering—that is, proper planning, careful modularity, and unit testing. The popular Ruby figure who went by the name “Why the Lucky Stiff” makes this point about conflicting values between software engineering and hacking forcefully and whimsically in “This Hack Was Not Properly Planned.”

Here’s a [more recent](#) invocation of the Platonic Ideal of Good Code. In this comment, we can see that the values of software engineering are more tacit, and more problematic:

> Every piece of academic code I have ever seen has been an unmitigated nightmare. The sciences are the worst, but even computer science produces some pretty mind-crushing codebases.

> “Ender7” on HackerNews

“Ender7” is replying here to a thread about a recent [Scientific American story](#) that suggested scientists were reluctant to release the code they used to reach their conclusions, in part because they were “embarrassed by the ‘ugly’ code they write for their own research.” According to Ender7, they should be ashamed of their code. Ender7 goes on to say:

> So, I don’t blame them for being embarrassed to release their code. However, to some degree it’s all false modesty since all of their colleagues are just as bad.

Why is academic code an “unmitigated nightmare” to Ender7? Because it’s not properly following the rules of software engineering. Again, the rules of software engineering presumably work well for them. I’m not qualified to comment on that. But that doesn’t mean that those values work for other contexts as well, such as biology. In this example, software engineering’s values of modularity, security, and maintainability might be completely irrelevant to the scientist writing code for an experiment. If scientists take care to accommodate these irrelevant values, they may never finish the experiment, and therefore never contribute to the knowledgebase of their own field. The question, then, isn’t about having good values in code; it’s about which values matter.

We often hear how important it is to have proper grammar and good writing skills, as if these practices had no rhetorical dimension, as if they existed in a right or wrong space. But we know from writing studies that context matters. Put another way: like grammar, code is also rhetorical. What is good code and what is bad code should be based on the context in which the code operates. Just as rhetorical concepts of grammar and writing help us to
think about the different exigencies and contexts of different populations of writers, a rhetorical concept of code can help us think about the different values for code and different kinds of coders.

And this is how coding values are relevant to us in computers and writing. The contingencies and contexts for what constitutes good code isn’t always apparent to someone just beginning to learn to code, in part because the voices of people like Ender7 can be so loud and so insistent. We know from studies on teaching grammar and writing that the overcorrective tyranny of the red pen can shut writers down. Empirical studies indicate it’s no different with code. Certainly there are ways of writing code that won’t properly communicate with the computer. But the circle of valid expressions for the computer is much, much larger than Ender7 or Dijkstra insist upon.

To close, I want to share with you a bit of what might be considered very ugly code, a small Logo program I call, tongue-in-cheek, “codewell”:

```
3 / 6
```
This is bad code because:

- it is uncommented and hard to read
- it is in an old, seldom-used language
- it is baggy and has repeated statements that should be rewritten as functions
- it is not modular or reusable
- it is an “unmitigated nightmare”

If you run the code [on github here] in a LOGO interpreter, it looks like this:

So, in addition to saying my code sucks, you could also say this:
it could be used to teach people some things about functions and code
it is a start for a LOGO library of letters that might be kind of cool
it does what I want it to do, namely, make my argument in code form.

Let’s imagine a world where coding is more accessible, where more people are able to use code to contribute to public discourse or solve their own problems, or just say what they want to say. For that to happen, we need to widen the values associated with the practice of coding.
To Edsger Dijkstra, I’d say: coding values that ignore rhetorical contexts and insist on inflexible best practices or platonic ideals of code should be considered harmful – at least to the field of computers and writing.

We need to "widen the values associated with coding," says @anetv Word. #cwcon

Amanda Wall