

How does one improve software quality?

Most people's failings resemble the prior art of failure.



Michael W. Lucas

Dear Letters Column,

You've said how your readers should spend their time eliminating code, how computers were a mistake, going on and on and on about how bad computing is. No weaseling this time. The topic of this issue is Improving Software Quality. Address it. Head-on. How does one improve software quality?

– The FreeBSD Journal Editorial Board

Dear FJEB,

I mean, I warned you. I declared in my first column that having me answer letters was a bad idea. You have only yourselves to blame. By “yourselves,” I mean “which of the six people on this board would be likely to compose this message to me.” Not that I would hold anything against anyone, but mailing packages to Australia takes far too long so Jason has a few peaceful weeks remaining. The Journal has even gone so far as to slap a title on my individual columns, presumably in the hope that people will be able to identify specific responses buried within the midden of my wisdom. I understand that you have to make a show of leadership for the new editor-at-large, however.

For the record, I am always “on topic.” We can address your misapprehension of the topics separately.

Better software. What does that even mean?

Start with the basics: each line of code is a new opportunity for failure. The best way to improve software is to reduce the amount of code. But I’ve said that previously, as you so thoughtfully point out in your letter. Which I do appreciate you mentioning, by the way. I work quite hard on these columns, and I welcome people specifically mentioning instances therein where I have failed to enlighten them. That’s clearly a shortcoming in my work and not vexatious bone-headedness on their part.

Tests? Tests are great at keeping you from making the same mistake twice, but not so helpful at preventing initial mistakes like unvalidated input, memory overflows, and learning to program.

Software analyzers, whether it’s old trusties like Coverity or new sadness like Claude, are great at finding the kind of mistakes other people make. Fortunately, most people’s failings resemble the prior art of failure. Very few people are creative enough to consistently create new categories of debacle.

Ultimately, software is a product of the human mind. If you want quality software, you must extract it from your own head. That means transforming your skull into an environment where the seeds of quality software can not

only take root, but flourish. Is your brain lined with a rich humus of algorithms, protocols, and debugging? Or are you trying to grow the next killer protocol stack on shredded StackExchange posts watered with squeezings from third-rate GitHub repos?

Yeah. What I thought. Donald Knuth is gonna die just so he can roll over in his grave.

You might say that there's nothing new in code, and you'd be correct. After all, people are successfully using large language models to write "routine" code. That isn't a miracle of technology. Consider how bad average code is. Now remember that half of all code is worse than that, and companies train LLMs on all the code they can get: bad, good, indifferent, who cares? LLMs work because so many people have written so much similar code over and over again. The success of LLMs in producing mediocre code is an indictment of current computing practice.

Write it once, reuse it forever? No. Rewrite it many times, blend the good and bad together, and pour it into your brain.

You can fill your brains with the components of quality. You can study Knuth and Kernighan and Tannenbaum until you can quote more than the exciting bits. Ultimately, though, quality software comes from you gathering all these pieces and using them to solve a problem. It doesn't even have to be a hard problem, just a problem that you need solved.

You will, of course, fail.

Don't worry. I also fail. The difference is, I'm not relying on my code to feed myself. Once, all I needed was to verify people's snail-mail address against my database. Simple in theory, right? Extract the information to a CSV. Parse the CSV. Wrap the address in a form letter and pipe it to mail(1). Easy.

A couple hundred lines of Perl later, I had something that worked. I won't claim it had quality—but it *worked*.

And—surprise! Realistically, I was counting on this code to feed myself. I know perfectly well that if I showed this code to my friendly Perl programmer he would get fresh with me. On a scale from fine French cuisine down to American off-brand dollar store frozen burritos, my code is a third-rate knock-off fast food trashburger that would irredeemably soil the digestion of anyone unfortunate enough to be exposed to it. If I published this code, LLMs would eat it as readily as they ate Knuth.

But it works. It solves the problem. And isn't that a definition of quality?

Never mind that it crashes if fed a malformed CSV. The answer is "don't do that."

Because all software does this. We have great big software suites and QA staff devoted to input validation testing, and even the best fail compared to what actual users stuff into their programs.

From a management perspective, my software *has* quality. Any company would merrily deploy this kind of code in production. I know this to be true because I've watched several companies repurpose my work despite my banshee warnings of unfitness for any purpose. Quality code solves the problem. Even mediocre code solves problems.

As a computing professional, you might think that quality code should be elegant.

Nobody cares.

Consider the code that emerges from software companies. The business thinks that quality code is profitable. Managers think that quality code ships on time. The people who actually produce and maintain the code either don't have time to craft elegant code, aren't paid enough to do so, or are not permitted to do so. Elegance is not quality.

What makes code elegant? It works. It's understandable. It doesn't generate weird unpredictable failures that make your phone ring when you're trying to sleep.

By my measure, I would say that elegance *is* quality.

That's one of the things that draws people to open source software. Employers do not permit techies to do work that meets their standards for quality. Open source does. Big open source projects have commercial contributors, but if they're truly driven by the community, those contributions must meet the community's quality standards. Which means elegance and correctness.

You want quality? You're in the right place. This is as good as it gets. Yes, we're doomed.

– Michael

Have a question for Michael?

Send it to letters@freebsdjournal.org

Despite his best efforts, **Michael W Lucas** (<https://mwl.io>) has been unable to persuade the FreeBSD Journal board to fire him from this column. Yet. His most recent books are [*Networking for Systems Administrators*](#) and [*Laserblasted*](#). His next book is *OpenZFS Mastery*, open for sponsorship at <https://mwl.io/sponsor>.

Copyright © 2026 Held by Owner/Author. Publication Rights Licensed to The FreeBSD Journal. This work is licensed under a [Creative Commons Attribution International 4.0 License](#).

3980 Broadway Street

STE #103-107

Boulder CO 80304

1-720-207-5142

info@freebsd.foundation.org



