

Read the Tea Leaves SOFTWARE AND OTHER DARK ARTS, BY NOLAN LAWSON

16 AUG

You can just choose how many bugs you want now

Posted August 16, 2026 by Nolan Lawson in software engineering. Tagged: AI. [Leave a Comment](#)

There's a bizarre aspect of AI coding that I've been trying to put my finger on, and I think it's this: you can basically just decide how many bugs you want your software to have now.

We discovered this first with security, because of course security bugs are the most non-negotiable ones. But I think once the vulnpocalypse (<https://en.wiktionary.org/wiki/vulnpocalypse>) is over, we'll start to turn our attention to other types of bugs: correctness, performance, accessibility, reliability, etc.

Some of us are already doing this. For example, I find myself spending a lot of time these days in code review, using tools like my triple-agent code review skill (<https://gist.github.com/nolanlawson/4150b0ca9640654c256b324fac0d5253>) as well as Geoffrey Litt's explain-diff skill (<https://www.geoffreylitt.com/2026/07/02/understanding-is-the-new-bottleneck>).

My experience is that, in a complex system, you can basically find as many bugs as you ask the agents for. If you get tired of tackling bugs in the PR itself, have no fear: the agent will also find plenty of preexisting bugs for you to spend time on. The question is just when you want to stop and call it "done."

Of course the bugs are not free to fix. There are still many tradeoffs to consider: lines-of-code versus likelihood that the bug will actually occur, the risk of introducing new bugs in a complex solution, the cost of making the code harder to understand for future reviewers or agents, etc. But the *finding* of the bugs has become nearly free, and AI agents are also capable of finding very subtle, intricate bugs that otherwise could have flown under the radar for years. What we do with this situation is the interesting question.

As many have noted, it doesn't seem like the overall polish of software has increased since AI coding became a thing. If anything, there is just more junk and shovelware out there, of dubious quality. I think this demonstrates that, although our ability to find new bugs has skyrocketed, our overall tolerance for bugs has not changed. There are still plenty of winds blowing in the opposite direction:

- The preventable problem paradox (<https://medium.com/@shreyashere/why-our-leaders-fail-us-and-then-save-us-the-preventable-problem-paradox-872614ec3293>): if an incident occurs and you swoop in to fix it, you're a hero. If you prevent the problem from ever occurring in the first place, then nobody knows you did anything.

- Related: the pressure inside many software orgs is to keep shipping visible results, not to fine-tune something that already “works.” With AI coding this is magnified: management often assumes that 10x productivity means 10x more visible features and apps.
- Laziness: one of the classic virtues (<https://threevirtues.dev/>) of a programmer, this time working against us. I find myself mentally exhausted after slogging through the umpteenth AI-generated bug report, which requires me to carefully think through intricate aspects of the system and weigh the pros and cons of fixing it. I imagine many of my peers in the industry have just tuned out AI code reviews or only focus on the most critical findings.

Avoiding epicycles

There are a few ways we can approach this problem, though, that don’t require unending toil. One way is to set up the agent on a loop, e.g. “do a code review, fix all critical/high/medium issues, then repeat.” I find this can work, but it has a tendency to create lots of epicycles (https://en.wikipedia.org/wiki/Deferent_and_epicycle).

If you’re not familiar with the concept: in the pre-Copernican¹ model of the solar system, ancient astronomers “fixed” miscalculations in the planets’ orbits by simply adding more circles to their movement. This improved the accuracy of the predictions, but at the cost of making the overall model more complicated. Obviously just saying “the earth moves around the sun” greatly simplifies the whole thing, but first you need the insight to make this simplification possible.

I’ve found that AI agents are pretty bad at such dramatic simplifications (in other words, “LLMs can’t jump” (<https://openreview.net/forum?id=klU4737opt>)). They will happily build one epicycle per bug until the code is a spaghetti mess. So a valuable part of AI code review is still to ask questions like “How can we make this simpler?” and “Is there a fundamental flaw with the codebase that we should fix before we tackle this class of bugs?”

Another technique that works well is to have good tests. (Easier said than done!) For example, when I was playing around with vibe coding the W3C IndexedDB API (<https://nolanlawson.com/2026/01/31/building-a-browser-api-in-one-shot/>), it became pretty clear to me that an agent could just grind through the test suite, and if it got close to 100% then I could be reasonably certain to have a bug-free implementation. But the only reason this works is because the Web Platform Tests (<https://github.com/web-platform-tests/wpt>) are a phenomenally good test suite, honed by years of independent browser implementers discovering odd bugs and adding test cases for every unlikely scenario you can think of. Most companies, in their first-party codebases, could only dream of such a test suite.

I can imagine, though, that if you’re building a system from scratch, and especially if your goal is to reproduce the output of an existing system, then you can get pretty far by just putting all your effort into the test suite and then letting the agent go nuts on the rest. PGRust seems to be having some success with this (<https://malisper.me/postgres-in-rust-regression-suite/>).

A third technique is to just simplify your system design so that whole classes of bugs become impossible. For example, I've long been an advocate for multi-page apps (MPAs) over single-page apps (SPAs) (<https://nolanlawson.com/2022/05/21/the-balance-has-shifted-away-from-spas/>), just because, with MPAs, entire bug categories simply don't exist: breaking the back button, losing scroll state, leaking client-side memory, improper accessibility during page navigations, etc.

Of course you lose some power with a simpler system versus a complex one, and maybe a reasonable answer is to deliberately choose a more complex system while also just fixing all the bugs. I feel though that this would still have a tendency towards epicycles, and I would much rather read (or debug!) a codebase built on simpler principles rather than one built on complex ones, even if they both have the same overall bug posture.

Conclusion

It's become cliché to note that we're in unprecedented times, and that everybody is figuring out what exactly software engineering is supposed to look like when robots can do a good chunk of what used to be "the job." And yet, it still remains worth saying. Whatever I wrote in this blog post may become outdated in a matter of months, and the next 5 AI-related articles you read on Hacker News will probably argue 5 different opinions. It's a cacophonous mess, and I have low confidence that I've figured out all the answers.

What I've defaulted to is focusing on the short term: i.e. what are agents good at today, and where can humans still provide some value. Some people are running with the assumption that all concerns of code quality, complexity, and maintainability will be swept away someday by agents that can easily manage whatever baroque legacy system they're handed. That may end up true, but I'm not going to bet on it because I haven't seen it yet. For now, I'm still concerned about things like the DRY (https://en.wikipedia.org/wiki/Don%27t_repeat_yourself) and KISS (https://en.wikipedia.org/wiki/KISS_principle) principles, keeping a working theory of the code in my head (ala Peter Naur (<https://pages.cs.wisc.edu/~remzi/Naur.pdf>))), and trying to steer the agent toward better code quality.

I do think it's interesting though, that we have a much greater ability to tackle more and more subtle bugs than we ever had before. Maybe this will lead to a reliability renaissance, or maybe it will lead to the same overall bugginess, just with more apps and more features in each app. I know that my personal preference is for greater software craftsmanship, but it remains to be seen how the software industry as a whole will step up to this challenge.

Footnotes

1. Technically, Copernicus's system still had epicycles (https://en.wikipedia.org/wiki/Deferent_and_epicycle#As_an_example_of_bad_science), and only Kepler managed to get rid of them. I think in some ways this makes the analogy stronger: in software, a re-

architecture sometimes isn't obviously better right away, and only shows its value over time.

This site uses Akismet to reduce spam. [Learn how your comment data is processed.](#)

Blog at WordPress.com.