

Building software with AI is cheap. I think keeping it alive is not.

An opinion piece

Kabui, Charles

2026-10-28

[Read at ToKnow.ai](#)



Building software with AI is cheap. I think keeping it alive is not.

Cheap to build. Costly to keep alive.

77%

agent runs: code eroded

+49%

pull request review time

October 26, 2026

ToKnow.ai

I think building software with AI is already cheap, and rebuilding it is cheap too. What I doubt is that anyone can keep it alive for years without paying a real cost. My guess is that the cost lands on people and money, not on the code.

Nicholas Carlini at Anthropic had 16 Claude agents write a [100,000-line C compiler](#) over nearly 2,000 sessions, for just under \$20,000. It builds Linux 6.9. But he wrote the tests himself, used GCC as a known-good reference, and said the checker has to be nearly perfect or Claude “will solve the wrong problem”.

Copying works for the same reason. A developer used Claude Opus 5.5 to build [ArtCraft](#), free apps that mirror Photoshop and other Adobe tools, working from public documentation and observed behaviour. They are labelled early alpha, but they exist. A finished product holds still, so there is something to compare against.

Keeping a product alive means dealing with things that are not in the code. An ISP throttles an endpoint. A user’s router is faulty. One API key is shared by so many people that it hits a global limit, so the product works for some and fails for others. A model treats every report as a bug to fix with code. It does not know the budget, or that a whole class of bugs is about to be dropped. Someone has to tell it, and that person must be skilled enough to judge.

Here is a real case. The sync tool rclone ships a Google Drive client ID that all its users share. Its maintainer wrote that the project is many hundreds of times over the free quota, and that Google charging for it could cost [“millions of dollars per year”](#). No code fixes that, it is a bill and a negotiation.

Even when the cause is in the code, I think an unsupervised agent patches the hole it can see. A person knows that if scenario “A” fails, then scenario “D” should fail too. If scenario “D” does not fail, then fixing scenario “A” did not reach the root cause. A study of 7,955 agent runs found that agents [“may focus on surface-level symptoms”](#) when they locate a bug.

Passing tests is not necessarily effective. In one study, the SWE-bench benchmark counted [7.8% of all patches](#) as correct even though they failed the developers’ own tests. When [METR](#) had maintainers review agent patches that passed the tests, their merge decisions came out about 24% below the automated score. Carlini hit the same wall: late in his project, new features kept breaking old ones until he built a continuous integration pipeline to stop it. And as [Addy Osmani](#) puts it, you cannot write a test for behaviour that nobody thought to specify.

I also think the patches pile up. The code gets bigger and messier, and the model spends more of its context window just reading it, so it has less left for understanding. In [SlopCodeBench](#), where agents keep extending their own code, structural erosion rose in 77% of runs. The agents’ code ended up about twice as eroded and 2.3 times as verbose as 473 open-source Python repositories.

If AI wrote all the code, and no human ever held the design, then no human can supervise it. Peter [Naur argued in 1985](#) that a program is really a theory held in the programmers’ heads. Someone who has that theory can explain why each part is what it is. He wrote that programs decay when they are modified “without a proper grasp of the underlying theory”. Osmani calls the gap between the code that exists and the code any person understands [comprehension debt](#).

Writing is fast and reading is slow, so that gap widens. A [Harvard study of 718 firms](#) found that AI agents raised lines of code by about 30%. Yet the time from opening a pull request to merging it rose 49%, and firm-level output did not rise by a significant amount.

If a human cannot step in, the only thing left to edit AI-written code is another AI, and probably a smarter one. That makes the lab your landlord, and a landlord sets the terms. OpenAI told developers it will [remove GPT-4o from its API on 14 January 2027](#). GitHub [moved Copilot to usage-based billing](#) on 1 June, and said that agent-heavy use had made its old pricing unsustainable. So holding a cheap model still is not something an API customer decides. Only a model you host yourself stays cheap.

If better code needs smarter models, and stronger models cost more, then I think code will eventually become a moat again. This is because only people with enough money can afford the strong models, and the gap in power and money grows larger. The people who can pay will have the best code. Eventually, ideas and skills will not be enough, and you may need money just to begin, because nobody wants a half-baked MVP. If a handful of assistants can make any small tool on request, there is less reason to install the other thirty apps on your phone, and an idea becomes one sentence in someone else's prompt.

In short, I expect job losses and the rise of super apps: users gain the freedom and convenience of shaping their experiences to their preferences in a single app, so they have no need to install or pay for any other app. Meanwhile, product owners whose apps are not already on users' phones are shut out, which widens the gap in wealth and power.

Disclaimer: For information only. Accuracy or completeness not guaranteed. Illegal use prohibited. Not professional advice or solicitation. **Read more:** [/terms-of-service](#)