AI is removing the middle class of software engineering

11 August 2026

Follow me on X Discuss on Hacker News

It's 2020. You're the most senior person on your team, in charge of code quality and architecture. You've set up good engineering practices, you thoroughly review PRs from people who are less experienced than you and work hard to maintain a healthy codebase.

Then at some point, you go on holiday. When you come back, the codebase is a mess. Everyone merged each other's PRs without really paying much attention, someone added a bunch of new tables to the database to denormalise it because it was easier and they added serverless or Kafka to the stack without any solid evidence that they needed either.

It's okay. You can fix this.

Fast forward to 2026. You haven't been on holiday. It's just a normal Monday morning. You make yourself a nice coffee, open your computer and find yourself with 7 PRs to review. You open the first one: +24586 −3938 lines, accompanied by some AI-generated description of what they're supposed to do. Somehow, your team has made more changes since Friday than they used to make while you were away for a few weeks.

AI removed the speed limit

AI makes projects with weak engineering culture fail much faster.

There used to be a time when people sat down and talked about how they'd do something. Now they can just prompt an agent for a few hours and open a PR.

The most tragic aspect of this way of working is that, to the untrained eye, it works.

If you pull the branch and test it, you'll probably get something somewhat functional. So what do they do? They keep going. Again and again. Until the project reaches a point where no one knows how anything works.

Just like someone buying a new luxury car on a credit card. You don't see the debt. You just see the car that looks great.

But then users start to report a weird bug. It's the 4th time your team has been trying to fix it. I mean... asking AI to fix it. Unfortunately, it seems like not even Fable can figure it out.

You go talk to the person who worked on this feature.

— "So where does the data come from?"

— "Hmm... actually I don't know. Let me ask Claude."

You sit next to each other watching an endless wall of text appear on the screen. Neither of you has any idea whether any of it is true but Claude seems very confident.

"Let's just turn on and ask it to double-check?"

This one will take a while. You start talking about the latest drama on X.

You finally get an answer back.

— "Does this make any sense to you?"

— "I'm not sure."

— "Didn't you build this like... last week?"

Silence.

This project has become so convoluted, with so many layers and services, that no one on your team could possibly start to understand what's going on.

So, what do you do?

Fixing it would require such a colossal amount of work that it would be impossible to even start justifying it to anyone in management.

And what are you even thinking about? It would end up in the exact same state again in just a few months anyway.

— "Let's just ask Claude to fix it."

— "Okay. I'll create a loop and goal so it doesn't stop until it's checked that everything works."

— "Sounds good"

— "Actually, I ran out of Fable usage for today so I'll run it tomorrow"

You grab another coffee and walk back to your computer. You now have 13 PRs left to review. You see something you don't quite understand, so you message the person who wrote it.

— "Why are we doing this here?"

They send you a link. It's a Claude conversation.

Somewhere in that conversation, buried between Claude confidently recommending one architecture, apologising, changing its mind, your coworker asking it to reconsider again and another 15 rounds of changes, is apparently the design decision behind this code.

— "Which part should I read?"

— "Probably all of it."

Does this sound familiar?

Whenever I talk about this, someone eventually tells me that nobody ever fully understood large systems anyway. It's true.

You were never expected to understand every service and every database. But at least someone did and would explain it to you.

Now they ask an LLM because they don't actually know themselves.

You can't afford bad engineers anymore

In every team, there are competent people who make the project possible. There are also people who essentially make it harder for everyone else. And now anyone can produce more code in a day than they used to in a year.

In the story above, everyone is failing:

- The engineer opening a 25,000-line PR should have stopped the agent long before it got there. They should have understood what it was doing, broken the work into smaller pieces and questioned every new abstraction it introduced.
- The person reviewing it should have refused to review something that large instead of giving in.
- The person adding Kafka should have been able to explain exactly why it was needed.
- The person who built the feature should have been able to explain where the data came from without sending a link to a Claude conversation.

But what's the problem then? Just use AI to fix it. Well, it's not that easy...

Before anyone jumps on this, none of this means technical debt is always bad. The important part is that you know it's a shortcut.

Anyway, reverting a bad decision is hard. Very hard.

For example, how long would it take an LLM to add a bunch of tables and columns to the database? 10 minutes?

But once you start storing data there, you can't just remove them. You have to come up with a migration plan, make sure you don't disrupt the system because people are paying to use this every day. You have to think about what you'll do if the migration fails. Make sure you don't end up with orphaned foreign keys. It's just so much harder to fix. Even with the best model you can get.

And while you're fixing it, more PRs keep coming in. More code, more abstractions, more decisions. A person can generate 20,000 lines of code in an afternoon, but you still have to sit there and understand what those lines actually do.

By the time you've untangled one bad decision, five more have been merged.

The new AI economy

Of course, bad engineers were always a liability.

It has been like this for decades, well before OpenAI or Anthropic existed. Bad decisions compounded, unnecessary complexity accumulated and teams ended up maintaining systems nobody really understood.

The difference is that there used to be a limit to how fast you could do it.

Today, implementation is cheap. You are paid to make good decisions. To build software that will scale while managing complexity.

Ask yourself why companies are paying six-figure salaries for engineers in London or San Francisco in the first place.

If all they needed was someone who could turn a specification into working code, why were they paying that much when they could already get it done cheaply elsewhere?

Why are the tech companies claiming that "software is solved" still paying top salaries to attract the best people they can?

My bet is that AI pushes salaries further apart. To be employable, there's a bar you have to clear and that bar is whatever the current best model du jour can do.

Good engineers have become more valuable because AI lets them move much faster. They don't need as many people around them just to do the implementation work anymore.

At the same time, bad engineers have become much more expensive to hire.

I wrote about this before when I said the vibe coder career path is doomed.

You need to contribute beyond what everyone already gets by giving an agent a prompt.

If you lack the judgment required to evaluate the LLM's recommendation, asking for more judgment doesn't solve the problem.

At some point, someone still has to know what is going on. And that's the most valuable person on the team.

The people who don't will become much cheaper to hire or get replaced entirely while the money gets funnelled towards an increasingly smaller number of people who can actually be trusted.

I don't think this is going to be limited to software engineering either. I believe the same thing is going to happen across most knowledge work. AI will make the best people much more productive and the bad ones almost impossible to hire. Before, there was a good chance someone would catch their bad decisions before they went too far. Now they can make changes faster than anyone around them can realistically review or understand them.

Answers to the most common objections

▸ **"Bad engineers always existed"**

▸ **"Just fix your process"**

▸ **"You are just anti-AI"**

▸ **"More output means more productive"**

▸ **"Pushing back just makes toxic"**

▸ **"Just AI output is like assembly from a compiler"**

▸ **"We ship 99% AI-generated code and it works"**

▸ **"Users don't care. It's just a CRUD app. Ship it and get paid"**

▸ **"What about junior developers?"**

▸ **"Using AI is just delegation, like a manager"**

▸ **"Not all technical debt is bad"**

▸ **"How do we make more senior devs if we don't hire juniors anymore?"**

▸ **"Will anyone actually care?"**

← Previous post
How to automate Instagram engagements with computer vision (and get banned)

Get new posts by email. No spam.

Email address Subscribe