The Hidden Tax on Senior Engineers

By Austen Tucker · 2026

The Hidden Tax on Senior Engineers

There's a study that should be pinned to the wall of every engineering leader's office.

Xu et al. (2025) looked at open-source projects before and after GitHub Copilot adoption. The headline finding was what you'd expect: more code got written. Peripheral contributors, less experienced developers newer to the project, generated more output. The overall productivity curve went up.

But here's the number that matters. After Copilot adoption, core contributors, the senior developers who maintained these projects, reviewed 6.5% more code. And their own original code output dropped by 19%.

That's not a rounding error. That's a structural reallocation of the most expensive, most experienced engineering time you have.

Let me say it differently.

AI tools made it easier for junior and peripheral contributors to write code. So they wrote more of it. But that code still needed to meet the project's standards. Someone had to review it. Someone had to check that the confident, fluent, syntactically perfect AI-assisted output didn't introduce subtle bugs, break conventions, or quietly duplicate logic that already existed elsewhere.

That someone was the senior engineer. The person who knows the codebase well enough to catch what the AI missed.

Every additional AI-assisted pull request that landed in the review queue was a pull request that a senior developer had to read, understand, and evaluate against their mental model of the project. The more the juniors produced, the more the seniors reviewed. The more the seniors reviewed, the less they built.

This is not a side effect. It's the mechanism. AI tools increase throughput at the bottom of the experience curve by shifting load to the top.

Think about what a senior engineer's time is actually worth.

Not in salary terms, though that's part of it. In terms of what they uniquely contribute. Senior engineers do the things that can't be delegated down: architectural decisions, complex debugging, the refactoring that keeps a codebase from collapsing under its own weight. They identify the problems nobody else has noticed. They say "we should not build this" when everyone else is excited about building it.

That work doesn't show up in lines-of-code metrics. It shows up in the absence of disasters. In the stability of a system over time. In the fact that the codebase is still comprehensible a year from now.

When you redirect 19% of that capacity into reviewing AI-assisted pull requests, you're not losing "code output." You're losing the architectural maintenance that keeps the whole system healthy. And you're losing it invisibly, because review work doesn't generate the same signal that building does.

Nobody notices when the refactoring pass doesn't happen. They notice six months later when the system is brittle and nobody can explain why.

The researchers framed this clearly: code produced after Copilot adoption "requires more rework to satisfy repository standards, indicating a potential increase in technical debt."

That sentence should haunt you. It means the additional output isn't free. It arrives with a maintenance cost attached. And the maintenance cost falls disproportionately on the people who were already doing the most valuable work.

It's a tax. A hidden one. The kind that doesn't show up on the dashboard but compounds quietly in the background until someone asks why the velocity numbers are great but the product keeps breaking.

Here's the resource allocation argument.

Most engineering organizations think about AI tools as a multiplier. Give everyone Copilot, everyone gets faster, total output goes up. The math looks clean.

But the math assumes that everyone's time is interchangeable. It's not. A junior developer's hour and a senior developer's hour are not the same resource. They produce different kinds of value. When the tool increases the junior's output at the cost of the senior's, the aggregate numbers improve while the composition of the work degrades.

You're getting more code. You're getting less architecture. More features merged. Fewer refactoring passes completed. More PRs closed. Less time spent on the work that keeps the next quarter from being a disaster.

If you're a CTO looking at a dashboard that shows a 15% increase in PRs merged after AI tool adoption, ask the follow-up question: where is that increase coming from, and what did it displace?

This is not an argument against AI tools. It's an argument for understanding what they actually change.

AI doesn't reduce the total demand on your senior engineers. It redirects it. The demand shifts from creation to verification. From building to reviewing. From the work they chose to do to the work that arrives in their queue because someone else's output increased.

The fix isn't to take the tools away from juniors. The fix is to account for the tax. Staff your review capacity deliberately. Protect senior engineers' building time the way you'd protect any critical resource. Measure review load as a first-class metric, not an afterthought. And stop treating "more PRs merged" as an unqualified win without asking what got pushed off the plate to make room.

The senior engineer who used to spend mornings on architecture and afternoons on features is now spending mornings on review and hoping to get to architecture next sprint. They won't. Next sprint will have even more AI-assisted PRs to review. The queue only grows.

That's the tax. It's real, it's compounding, and it's invisible to anyone who only looks at the top line.