THE NO-BULLSHIT GUIDE TO AI IN SOFTWARE DEVELOPMENT

What it can do. What it can't.
And what it will cost you to find out the hard way.

DAVID LOGAN · Your Software CTO · david@yoursoftwarecto.com

Published Q1 2026

> ## "CODING IS LARGELY SOLVED."
>
> *— Boris Cherny, creator of Claude Code, Lenny's Podcast, February 19, 2026. Cherny has not manually edited a single line of code since November 2025.*
>
> He is not alone. The same week, Spotify's co-CEO told analysts that the company's most senior engineers "have not written a single line of code since December." They generate code and supervise it. That's it.
>
> Neither of them is wrong about what they are experiencing. Boris built the tool. He owns the architecture completely. Every decision about what to build and why lives in his head. Claude Code executes what he already knows needs to happen. Spotify's senior engineers are doing the same: directing, reviewing, merging. The humans are still deeply in the loop. They just stopped typing.
>
> What both of them may be underestimating is how much of software engineering was never in the typing.

This paper exists because that distinction matters, and because the gap between their experience and your enterprise codebase is exactly where bad decisions get made. The experiences they describe are real. But they describe automation of implementation, not automation of software engineering. Those are not the same thing.

THE CLAIM ON THE TABLE

SOMEONE TOLD YOU AI WILL REPLACE YOUR DEVELOPERS.

Or that your competitors are already running lean engineering teams powered by agents. Or that developers who can't prompt their way to a feature are becoming obsolete. We went looking for documented proof of any of it.

This paper is not a strategy guide. It will not tell you what to buy or what to deploy.

What it will do is give you an accurate picture of what AI can actually do in software development today, what the evidence says about where it fails, and what your organization has to be before any of the gains are even available to you.

Most organizations are not there yet. This paper will tell you why, and what that means.

PART 1

WHAT THE RESEARCH ACTUALLY SHOWS

We searched. We filtered out every vendor study, every consultant case study, every productivity narrative dressed up as evidence. We were looking for one documented case, just one, of AI autonomously writing complex enterprise business logic in production. Not AI helping a senior engineer who already knew exactly what to build. Not AI generating code from a Slack message while a developer reviewed and merged it on their commute. Autonomous authorship of the kind of logic that makes a business actually work: pricing rules, compliance engines, transaction processing, domain-specific algorithms. Core business logic.

WE FOUND NONE.

Not at Goldman Sachs. Not at Stripe. Not anywhere. Two rounds of research, strict filters, zero vendor sources. Same conclusion.

What we found instead was a precise map of what AI **actually can do** in software development today. The picture is more limited than the hype, more useful than the skeptics allow, and entirely conditional on engineering decisions most organizations have not yet made.

The closest thing that exists is Stripe's internal "Minions" system, which merges over 1,300 AI-authored pull requests per week. It is explicitly limited to maintenance work: dependency management, flaky test fixes, documentation updates. Stripe's own engineers prohibit the agents from writing greenfield business logic. The human review gate is not a formality. It is, in their words, load-bearing.

One more thing worth saying plainly: the data does show that AI is reducing the number of junior developers companies hire. Early-career engineering roles have declined 13–20% since widespread AI adoption. That is real. But junior headcount declining is not the same as developers becoming optional. What the data actually shows is that the ratio is shifting: fewer people generating code, more senior engineers governing the output. The cognitive core of software engineering, including architecture, system design, business logic, and judgment under uncertainty, remains entirely human. AI handles more of the execution. Humans still own all of the thinking.

Before looking at where AI genuinely helps, we need to clarify the distinction that makes most of the hype narrative misleading. (Whether eliminating the entry-level pipeline is a long-term mistake for the industry is a separate conversation, and one worth having.)

The experiences people like Boris Cherny and Spotify's senior engineers are describing are real. But they describe automation of *implementation*: the translation of known decisions into working code. They do not describe automation of *software engineering*: the process of figuring out what to build, how to structure it, what

tradeoffs to make, and whether it actually solved the problem. Those two things are not the same. Implementation was never the hard part. AI has gotten very good at the part that was never the bottleneck in the first place.

A note on where the ceiling is headed. The reasoning engines powering today's AI tools are genuinely more capable than they were a year ago. Modern models now run a hidden deliberation phase before answering, drafting, backtracking, running code in sandboxes, and verifying their own work before producing a response. Recent benchmark results reflect this progress, though the picture is more nuanced than the headline numbers suggest.

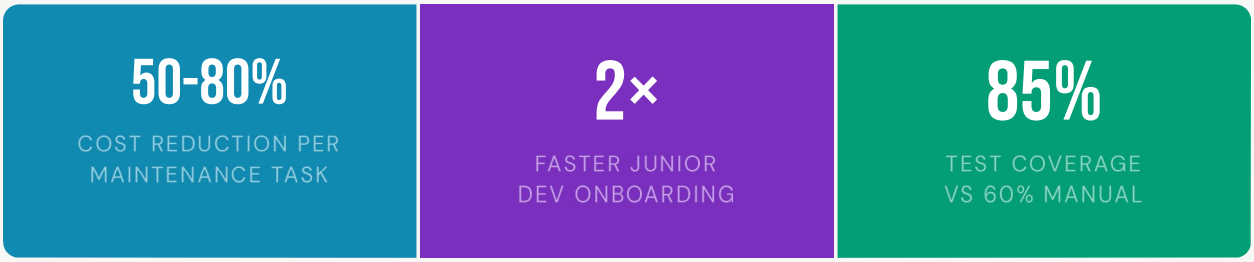
On the widely-cited **SWE-bench Verified**, a benchmark where AI attempts to resolve real GitHub issues from public, well-documented repositories, frontier models now successfully resolve around 80% of the issues presented. That sounds like near-human performance. But on **SWE-bench Pro**, which uses private proprietary codebases the models have not encountered during training, the same frontier models resolve only 23-46% of issues, depending on how the model is deployed. The public benchmark measures resolution rates when AI has probably seen similar code during training. The private benchmark measures resolution rates when it has not. That gap is the clearest available measure of autonomous enterprise coding capability as of Q1 2026. While the ceiling is rising, it is not yet high enough to change the argument. When it does change, the indicator will not be a benchmark score. It will be a documented production deployment. We are still looking for the first one. (Source: Scale AI SEAL Leaderboard and SWE-bench Verified, February 2026. Scores change; verify at swebench.com.)

PART 2

WHAT AI ACTUALLY CAN DO WELL

Once you separate implementation from engineering, a clearer picture emerges.

Today's AI is already very good at three specific kinds of work, and the economic case for each is real and documented. Each section below describes what works, followed by the preconditions that must exist before it applies to your organization.



Maintenance and Technical Debt

The most defensible AI ROI in software development today is not writing new features. It is handling the maintenance work that every engineering team knows matters and almost none actually stays on top of. Think of it as an automated operations layer: always running, never complaining, never deprioritizing a dependency upgrade because a sprint got crowded.

The workflow that works looks like this: a ticket arrives describing an observable failure. The agent writes a test from that ticket. The test fails. If it does not fail, something is wrong and a human needs to look at it immediately. The agent then traces the failure back through the codebase to the source, fixes the code, confirms the test passes, and submits a PR with the new test included. A human reviews and merges.

That same pattern applies to other well-scoped janitorial tasks: a test that fails non-deterministically due to a race condition or shared state gets diagnosed and stabilized. A deprecated internal API used in 40 places across the codebase gets migrated to the new one systematically. An incident post-mortem gets drafted from observability data in 15 minutes instead of 90. Dependencies that have quietly fallen three major versions behind, the ones nobody ever gets around to, get bumped, tested, and submitted for review continuously instead of accumulating for years until the upgrade becomes a six-week project.

The unit economics are real. Depending on task complexity and tooling, maintenance work handled by an AI-assisted pipeline typically costs **50–80% less than the same work done by a human engineer,** even after accounting for human

review overhead on every PR the agent produces.

Source: Engineering economics modeling based on SWE-bench resolution rates and fully-loaded U.S. engineering labor costs, 2024-2025. Illustrative range; actual savings vary by task type, codebase, and tooling. Current as of Q1 2026.

> ⚠ CRITICAL PRECONDITION
>
> This only works if your CI pipeline **has a trustworthy acceptance gate.** The entire workflow depends on a failing test meaning something real. If your test suite is sparse, unreliable, or disconnected from the code paths the agent will touch, the agent has no reliable way to know whether its changes worked or broke something. It either loops indefinitely or produces pull requests that cannot be trusted. This ROI category is not available to you until the test suite is. That said, for legacy organizations, generating AI-driven tests that describe what the code currently does, even imperfectly, is a valid first step toward that gate, not a reason to wait for perfection.

AI-Generated Testing

AI-generated test coverage is primarily a risk reduction and quality improvement tool, not a developer productivity play. When an AI-assisted testing framework is applied correctly, it catches defects that human testers miss, reduces the rate at which bugs reach production, and accelerates recovery when they do. In a 2025 study of a financial services application processing 10,000 daily transactions, an AI-driven testing framework detected 94.7% of injected defects compared to 83.3% for human testers. In one instance, it identified a security flaw that manual review had missed entirely, a flaw projected to cause a $50,000 breach if it had reached production. For PE-backed companies specifically, improved defect detection has an additional dimension: comprehensive, auditable test coverage is becoming a direct factor in regulatory compliance readiness and cyber insurability.

Source: Naqvi & Baqar (2025), AI-augmented testing framework study, financial services application, 10,000 daily transactions. Current as of Q1 2026.

⚠ CRITICAL PRECONDITION

How you use AI for testing determines whether you get a safety net or a false sense of security. If you hand AI your existing code and ask it to generate tests, it will test what the code *does,* not what it *should* do. A bug gets a passing test. The correct approach is test-driven: give the AI the specification or the ticket first, let it write the test before touching the code, and treat a test that does not fail immediately as a signal that something is wrong, not a sign that everything is right.

Developer Onboarding and Documentation

Junior developers reach productivity nearly twice as fast with AI assistance. Time to the 10th merged PR drops from 91 days to 49 days. Documentation that used to take weeks, and was usually never produced at all, can be generated and maintained automatically, achieving up to 89% coverage compared to the 20-30% typical of manual approaches.

Source: Developer Experience (DX) platform telemetry across thousands of engineers, Q4 2025; D3 framework study, 52 practitioners, weighted average productivity improvement of 26.9%.

⚠ CRITICAL PRECONDITION

AI accelerates code generation at the junior level. It does not automatically reduce the review burden on senior engineers. Handing out AI tools without addressing review capacity will increase the volume of pull requests arriving at the same bottleneck that already existed. The gains described above are real. Capturing them requires that the review side of the pipeline can absorb the volume the generation side is about to produce. Part 3 addresses what happens when it cannot.

PART 3

THE SENIOR BOTTLENECK

AI accelerates code generation at the junior level. That acceleration does not disappear into productivity gains. It travels upstream as volume. More code gets written. More pull requests get submitted. Each one requires a senior engineer to review it. The organizations that discover this after deployment rather than before are the ones that hand out Copilot licenses and wait to see what happens.

The bottleneck operates on two fronts. First, the volume problem: AI-generated code floods the review queue. Second, and separately, senior engineers working on complex legacy problems are themselves measurably slower with AI than without it. These are different phenomena with the same root: AI does not reduce the cognitive load of senior-level engineering work. It concentrates it.

+98%	+91%	-19%
PR VOLUME, HIGH AI ADOPTION TEAMS	REVIEW TIME INCREASE, HIGH AI ADOPTION TEAMS	SENIOR DEV SPEED ON COMPLEX TASKS

Sources: Faros AI telemetry, 10,000+ developers across 1,255 teams (2025). METR randomized controlled trial, Feb-Jun 2025, 16 developers, 246 real-world issues. Data current as of Q1 2026.

> *"AI doesn't fix a team. It amplifies what's already there."*
>
> *The DORA research program is the longest-running study of software delivery performance. The 2024 report found that as AI adoption increased, delivery throughput dropped 1.5% and delivery stability dropped 7.2%. The 2025 report found positive correlation with throughput in mature organizations, but confirmed that AI continues to worsen stability where engineering discipline is weak. Both reports reach the same conclusion.*

PART 4

THE ORGANIZATIONAL PREREQUISITES

Every one of the gains described above has a list of things that have to be true before it is available to you. Not eventually. Right now. Before you spend a dollar.

We are not the only ones saying this. Factory, a $300M AI software development company whose entire business is deploying agents into enterprise codebases, independently developed a five-level agent readiness framework. Their conclusion: most teams need to reach their own Level 3 tier, what they call "Standardized," before agents can be effective. Getting there requires exactly the prerequisites described below.

Do you trust your test suite?

Not "do you have tests." The real question is whether your team confidently deploys from passing tests, or whether a green CI build still sends everyone scrambling through manual QA. If your tests pass and your team trusts them, you have an acceptance gate. If passing tests still require a full manual QA cycle before anyone believes the build is safe, the pipeline has no reliable foundation to work from.

Is your codebase findable?

When a bug lands, can your engineers identify the relevant area of the code from the ticket alone, or does finding it require reproducing the failure first? An AI pipeline needs a starting point. It cannot read your entire codebase looking for one. A failing test provides that starting point automatically. A well-structured codebase makes the path from failing test to root cause navigable. A codebase where trading rules live in `market/jun2025/newstuff/logan/rules.java` inside a class named `MustHaveTradingStuff` is the answer to why your pipeline will fail before it starts.

Are your tickets written with observable specificity?

A ticket needs to describe behavior precisely enough that someone, or something, could write a failing test from it without ever talking to the reporter. "I click this button, I get X, I expected Y" is a ticket an agent can work from. "Something seems off with the checkout flow" is not.

For most organizations, meeting this standard requires a fundamental rethink of how problems get documented, not just better ticket hygiene. It means engineers, QA, and product stakeholders all need a shared discipline around describing observable behavior: inputs, expected outputs, actual outputs, and reproducible steps. That is not a tooling problem. It is a process and culture problem. The good news is that it costs nothing to fix except attention and consistency. The bad news is that most organizations have been writing vague tickets for years and changing the habit takes longer than anyone expects.

Does your staff have the capacity to build and maintain the harness?

The AI agent is not the product you buy. The infrastructure around it, the sandboxing, context injection, CI gates, cost caps, observability, is what makes it safe to run. Without it, agents generate runaway code, produce hallucinated refactors, and rack up API costs with nothing to show for it. That harness requires real engineering investment to build and maintain. It is not a vendor's problem to solve. It is yours.

Do you have a PR process that can absorb the volume?

Every AI-assisted pipeline produces pull requests. Lots of them. You need a PR process robust enough to handle significantly higher volume without becoming the new bottleneck. Clear ownership, defined review standards, realistic bandwidth. If your current PR process is informal or inconsistent, fix that first. The pipeline will expose those gaps immediately.

PART 5

THE READINESS SPECTRUM

These prerequisites define a practical readiness spectrum. The question is not whether AI is useful in software development. The question is where your organization sits on the line between "not yet" and "ready now." The tiers below describe four recognizable states. Most organizations will recognize themselves immediately.

One thing is true at every tier, including the bottom: individual developers using AI tools thoughtfully, for research, for drafting, for working through a problem, will see personal productivity gains. That has always been true and remains true regardless of where your organization sits. It is just not the same thing as organizational ROI, and it is not what the people selling you platforms are talking about.

TIER 1	TIER 2	TIER 3	TIER 4
Not Yet	Further Back	Close	Ready

TIER 1 — NOT YET

Old system. No meaningful test coverage. No documentation. Codebase that requires a guide just to navigate. Key people may have left or be leaving. This is not a technology problem yet. It is a human capital problem. No AI-assisted pipeline delivers organizational ROI here until senior engineers stabilize and document what exists. Have a direct conversation with your board about this. The cost of pretending otherwise is higher than the cost of the conversation.

TIER 2 — FURTHER BACK THAN YOU'D LIKE TO ADMIT

Legacy system. Sparse tests. Codebase navigable only by the people who built it. Architecture that lives in two or three heads. This is more common than anyone admits in a board meeting. The AI-assisted SDLC is not available to you yet. But AI's first job here is making the codebase findable and generating the documentation that makes everything else possible. That is real value. It is just not the ROI being advertised.

TIER 3 — CLOSE

The product works. Tests exist but are not fully trusted. Codebase is mostly navigable with some dark corners. Tickets are a mixed bag. PR process is informal. You are 6-9 months of foundational work away from reliable ROI. Start with test coverage and ticket discipline. The path is clear. It just requires doing the unsexy work first.

TIER 4 — READY

Reliable tests your team trusts. Findable codebase. Tickets written with observable specificity. Staff capacity to build the harness. A PR process that can absorb the volume. The AI-assisted SDLC is available to you now. ROI in 60-90 days if you execute well. If this is you, you already know it. You are probably already doing it.

THE CLOSE

THREE QUESTIONS WORTH ASKING

Before your next board meeting or vendor conversation, ask these instead of "are we using AI":

1. *Does your team deploy confidently from a passing test suite, or does green CI still trigger a manual QA scramble?*

2. *Can your engineers find the relevant code from a bug ticket alone, or does fixing anything require reproducing it first just to know where to look?*

3. *If your two most senior engineers left tomorrow, how long before anyone else understood the system well enough to change it safely?*

The answers tell you more about your AI readiness than any vendor demo, any tool assessment, or any consultant's maturity model. And they're free.

YOURSOFTWARECTO

David Logan · david@yoursoftwarecto.com