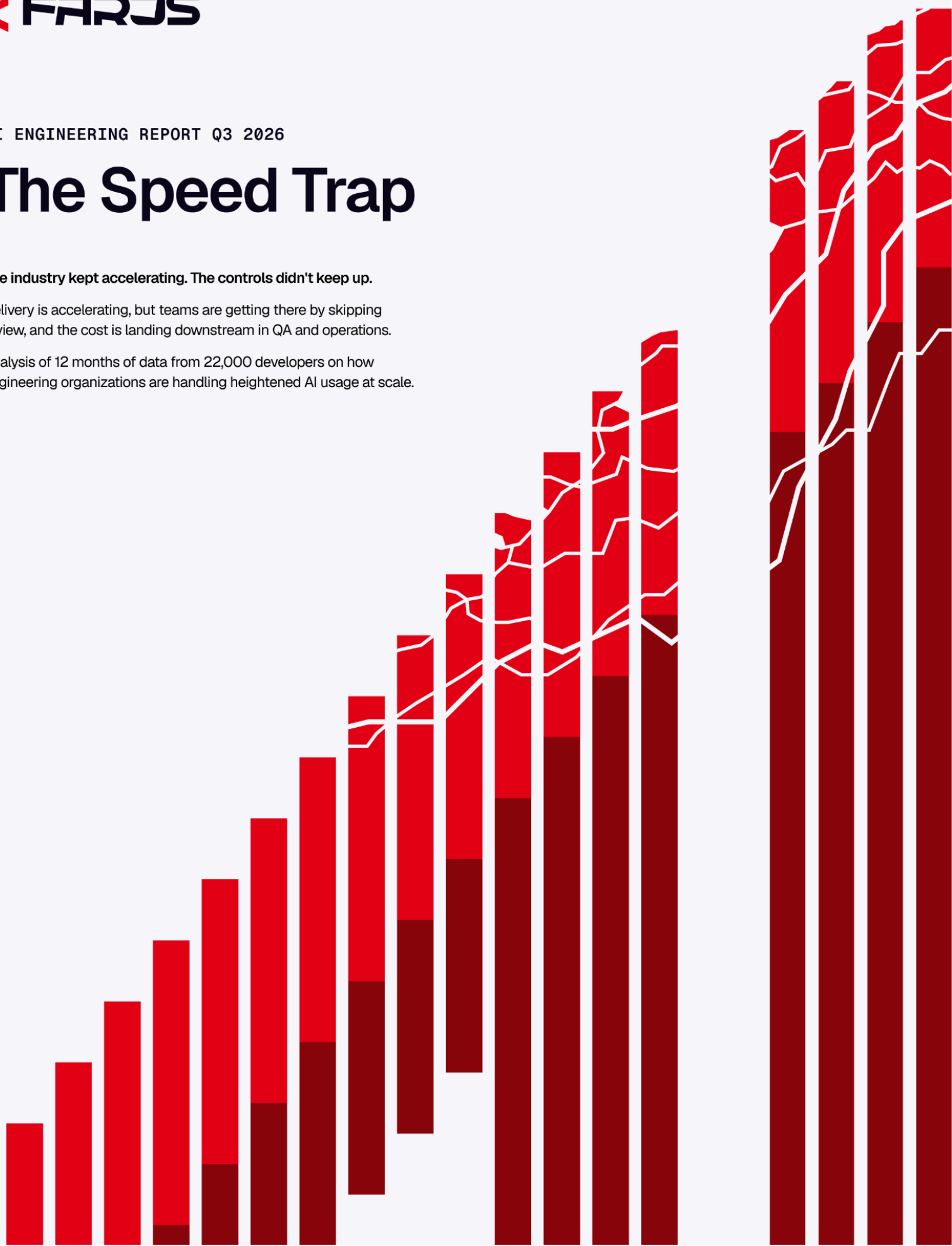


The Speed Trap

The industry kept accelerating. The controls didn't keep up.

Delivery is accelerating, but teams are getting there by skipping review, and the cost is landing downstream in QA and operations.

Analysis of 12 months of data from 22,000 developers on how engineering organizations are handling heightened AI usage at scale.



EXECUTIVE SUMMARY

The system is adapting. The strain is moving downstream.

Six months ago, we documented the Acceleration Whiplash: AI had increased engineering output faster than organizations could absorb it. Throughput rose, but bugs, incidents, rework, and workflow delays were climbing even faster.

This report tracks what happened next.

Some of the worst effects from that transition are beginning to ease. Incidents per pull request are up **14.5%**, compared with a 242.7% increase in our prior dataset. Code churn is up **71.6%**, down from **861%**. Deployment frequency has reversed direction, with high-AI-adoption teams now deploying **13.8% more**. Teams are getting better at turning AI-generated output into shipped software.

But that improvement reveals a new risk: **the speed trap**.

As organizations get better at moving AI-generated work through the system, they are also letting some of the controls slip. PRs merged without any human or agentic review are up **76.3%**, more than double the increase we reported six months ago. Time in QA is up **300.6%**. Monthly incidents are up **125.4%**, mean time to remediate is up **25.8%**, and backlog growth has reversed from shrinking to growing **42.8%**.

Each individual PR is now far less likely to cause an incident than during the transition period. But there are many more PRs moving through the system, and more of them are bypassing review. Per-change risk is stabilizing while total operational load continues to rise.

That is the *speed trap*: optimizing for faster code creation and delivery while the costs of verification, remediation, and operational load accumulate further downstream.

But there's a promising counter-signal. Agentic code review now covers **26%** of PRs. Teams that use it heavily see time to first review drop **10.7%** and change failures drop **24.4%**, the first intervention in our data that shows measurable benefits at scale.

The story of this edition is therefore less about whether AI makes engineering faster. It does. The question is **whether organizations are mistaking increased throughput for improved, sustainable performance**.

The next phase of AI engineering is about escaping that trap. Organizations need to rebuild these controls, the gates around AI-scale output, through policy, tooling, and agents before autonomous authorship increases the volume again.

Report's purpose

Our first report (July 2025) found the *AI Productivity Paradox*: heavy AI investment without matching gains in delivery. Our second (April 2026) found that paradox had sharpened into the *Acceleration Whiplash*: throughput rising while bugs, incidents, and rework rose faster still.

This report asks whether the whiplash has run its course. It hasn't, but it looks different, and is worth a closer look.

This report is for engineering leaders, CTOs, and the organizations advising them. As before, we let the data make the argument, while keeping editorial commentary anchored to the numbers.

Three things to keep in mind while reading this report:

- **These are independent snapshots, not a tracked longitudinal panel.** Each report samples a different dataset. Comparisons across editions show directional consistency, not precise period-over-period trends.
- **This edition covers a different phase.** Every team in this window (12-month period ending in August 2026) was already using AI, moving from established to more intensive use, while the prior edition measured an earlier transition. Expect smaller shifts as a result: a flat or improving metric here is a stronger signal, while a worsening one is a genuine warning sign. We also cannot fully isolate two concurrent trends: organizational maturation in AI use, and improvements in the underlying models.
- **We compared each team against itself.** For every team, we measured change between its own lowest-AI-adoption quarters and its own highest-adoption quarters, rather than comparing different companies against each other. We run through the full research methodology in this report's appendix.

FINDINGS

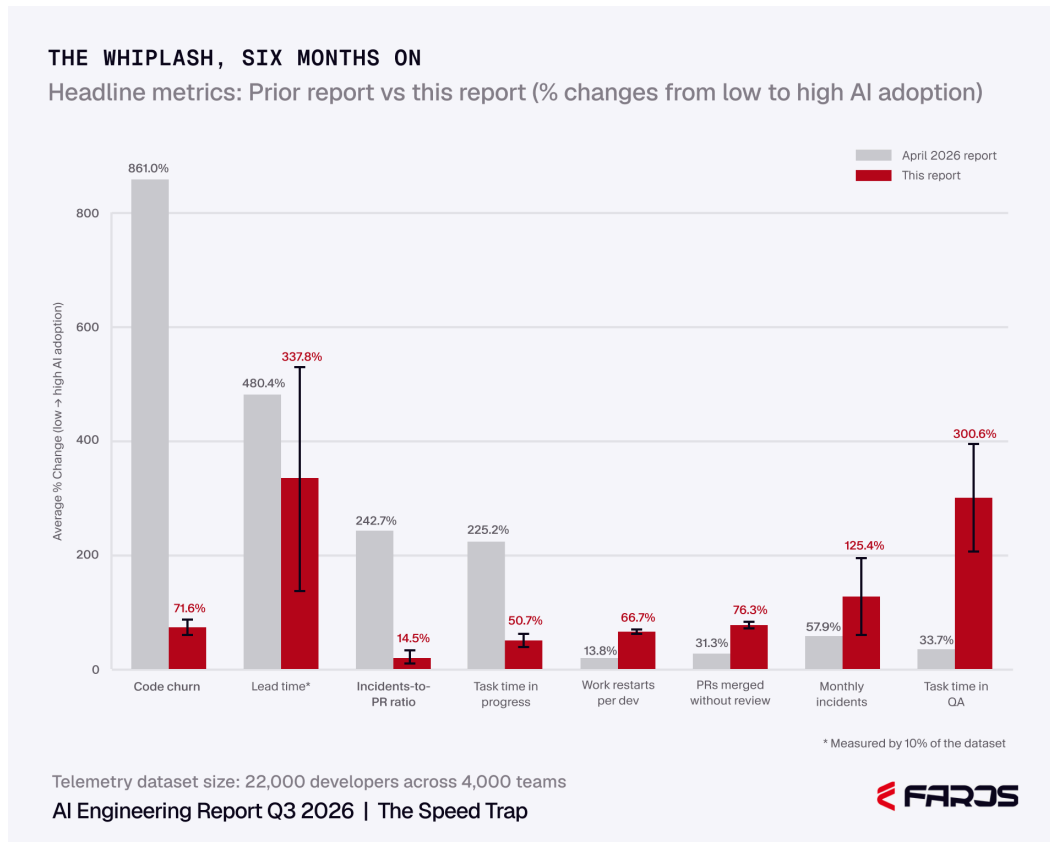
Less whiplash. Fewer controls.

In April, every downstream stage told the same story: volume up, quality down, gap widening. This dataset splits that story in two.

Two kinds of governance are moving in opposite directions. Spend governance, the caps and budgets on AI consumption, is tightening. Change governance, who or what reviews code before it merges, is eroding, and the pace is worth every engineering leader's attention.

The distinction matters because the two manage different risks. Spend governance controls what AI costs. Change governance controls what it produces. Organizations are getting the first one under control while the second one quietly slips: some teams adapted by building better scaffolding, like agentic review and context provisioning; others adapted by cutting the cheapest friction available, which is review itself.

That same split, some things stabilizing while others erode, shows up across the rest of this dataset too. Some patterns are holding steady from the prior report: review load continues to decentralize, PRs keep getting larger, review comments keep multiplying and lengthening, and time-in-workflow-stage metrics remain elevated with high variance. Others have moderated substantially, including code churn, the incidents-to-PR ratio, task time in progress, lead time, and context switching on the PR dimension. And a few have worsened despite the higher baseline: PRs merged without any review, work restarts, mean time to remediate, monthly incidents, backlog growth, and time in QA.



Here's how we break down the findings below:

1. AI Adoption
2. Developer Cognitive Load
3. Throughput
4. Code Size and Spread
5. Pre-Merge Code Quality
6. Flow and Efficiency
7. Code Quality in Production
8. Agentic Review

FINDING #1: ADOPTION

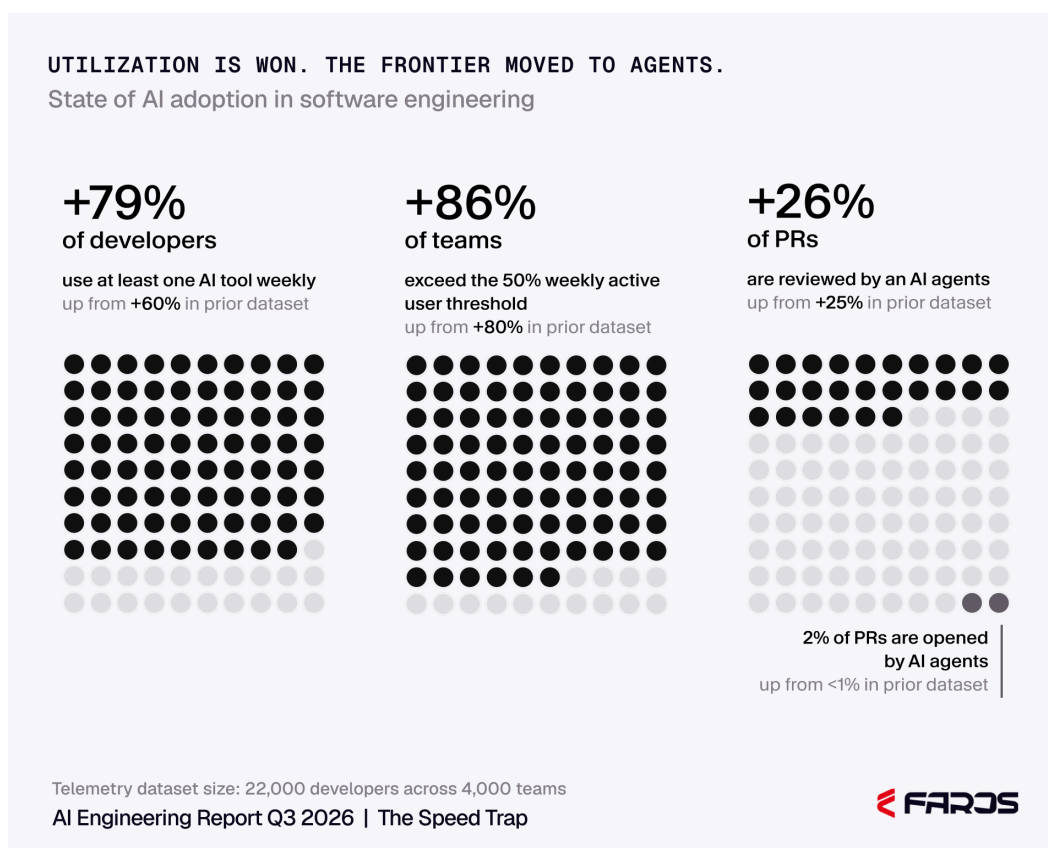
The rollout is done. Agents come next.

Acceptance rates inflected sharply this year. AI review went mainstream. AI authorship found its first breakouts.

The AI adoption race we described in April, in which organizations focused on driving utilization among developers who had licenses but were not using them, is largely over. Weekly active usage is high and still climbing, and the interesting shift now is the types of tasks developers choose to delegate to AI.

State of AI adoption in software development:

- **79%** of developers use at least one AI tool weekly (up from 60% in prior dataset)
- **86%** of teams exceed the 50% weekly active user threshold (up from 80% in prior dataset)
- **65%** acceptance rate of AI-generated code (up from 60% in prior dataset)
- **26%** of PRs are reviewed by AI agents (up from 25% in prior dataset)
- **2%** of PRs are opened by AI agents (up from <1% in prior dataset)



Three movements in this data stand out:

1. **The acceptance-rate inflection was synchronized.** Acceptance rates across the companies in our dataset moved together in the second quarter of 2026. Companies that had sat at 20–40% acceptance for years moved to 60–100% within a single quarter, with the shift happening across the cohort at roughly the same time. When adoption metrics move in lockstep across an industry like that, it suggests the tools themselves are improving dramatically. A generation of models and agent harnesses crossed a usefulness threshold, and developers everywhere responded in similar way at the same time.
2. **Agentic review went from novelty to norm.** In this dataset, AI review is now integrated into the workflow, with many companies running agentic review on over 50% of their PRs. This is the fastest adoption curve we have measured for any AI capability in the SDLC, and the first whose downstream effects we can now measure at scale.
3. **AI authorship remains rare, but it is no longer uniform.** For most companies, PRs opened autonomously by AI agents still account for less than 5%. But a vanguard has broken away: at the leading edge of our dataset, agents now open 13–14% of all PRs, and that share is climbing quarter over quarter. The gap between the median and the frontier is widening. What happens to the vanguard's quality metrics over the next year is, in miniature, the question the rest of the industry is about to face.

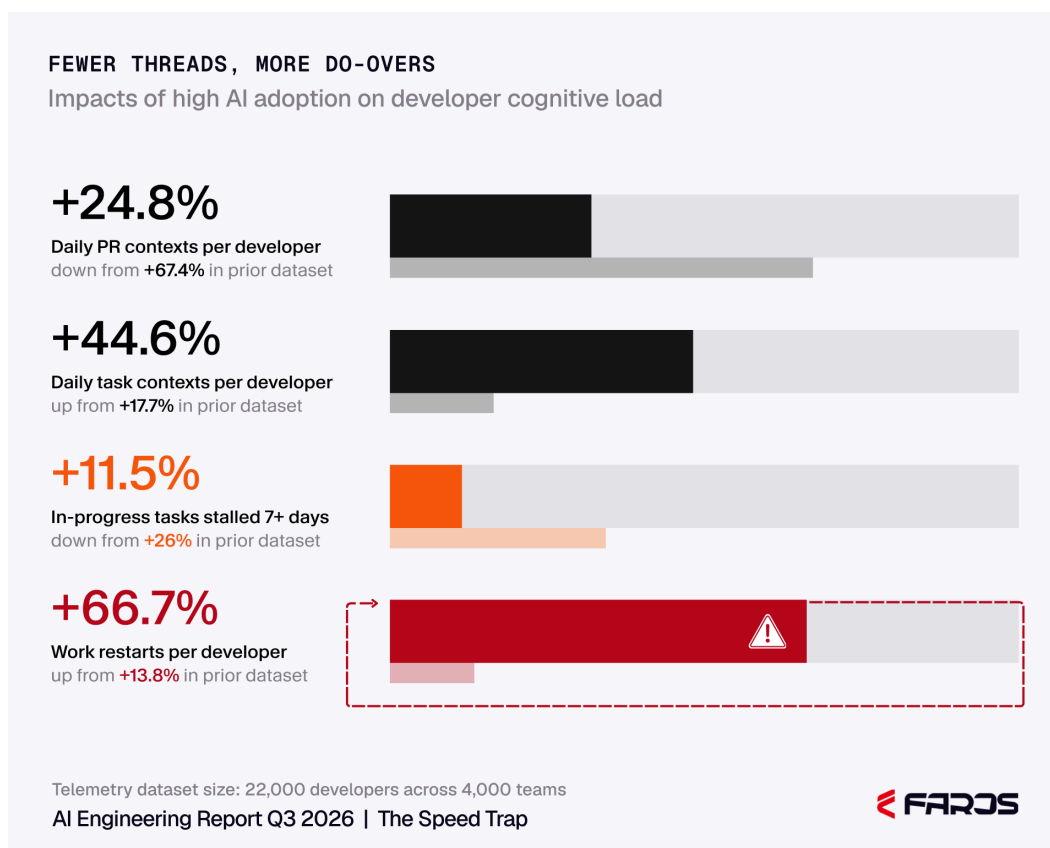
FINDING #2: DEVELOPER COGNITIVE LOAD**Thrashing changed shape.**

Developers juggle fewer parallel PRs than the peak. They restart far more work.

In the prior report, the cognitive load story was about parallelism: developers touched 67.4% more PRs per day and opened more threads than they could finish. In this dataset, that specific pressure has eased and been replaced by something arguably worse.

Context-switching metrics:

- **+24.8%** daily PR contexts per developer (down from +67.4% in prior dataset)
- **+44.6%** daily task contexts per developer (up from +17.7% in prior dataset)
- **+66.7%** work restarts per developer (up from +13.8% in prior dataset)
- **+11.5%** in-progress tasks with no PR or activity in the past 7 days (down from +26% in prior dataset)



The parallelism tax is moderating. PR contexts per day are still elevated, but far below the peak of the adoption transition, and stalled work (tasks started and then abandoned for a week or more) has also decreased substantially. Developers appear to be getting better at managing the fan-out that AI makes possible.

Meanwhile, the restart tax is exploding. Work restarts (tasks returning to in-progress after moving to another stage) are up 66.7%, nearly five times the increase in our prior dataset. A restart can happen after review, QA, or even post-deployment. Before AI, a restart was as likely to signal context switching as defective work. Under high AI adoption, we believe the dominant scenario is now the latter: the authoring stage produced something that did not survive review or QA.

A restart is the most expensive kind of context switch. The developer has to reconstruct where they were, what they were trying to solve, and why they made the decisions they did. Then the second attempt burns another round of tokens on work that was already paid for once.

We read this shift as a direct signal about agent context quality. In April, we recommended watching restarts as an indicator that coding agents had taken the wrong path and work had to begin again. The data in this edition strengthens that interpretation considerably. As more authoring is delegated to AI, the dominant cognitive cost is increasingly the downstream discovery, during review, QA, or deployment, that the work went in the wrong direction. The fix remains upstream: better context provisioning before the agent begins, rather than costly and time-consuming manual correction after a restart.

Daily task contexts rising while daily PR contexts fall fits the same picture. Developers are supervising more units of work while directly manipulating fewer of them. That is what orchestration looks like in telemetry, and it offers a preview of the agentic workflows to come.

A restarted task is paid for twice: once in the tokens that produced the wrong output, and again in the tokens that produce the replacement, with the human time required to discover the error sitting on top of both. At a 66.7% increase, restarts are the largest identifiable source of duplicated AI spend in this dataset.

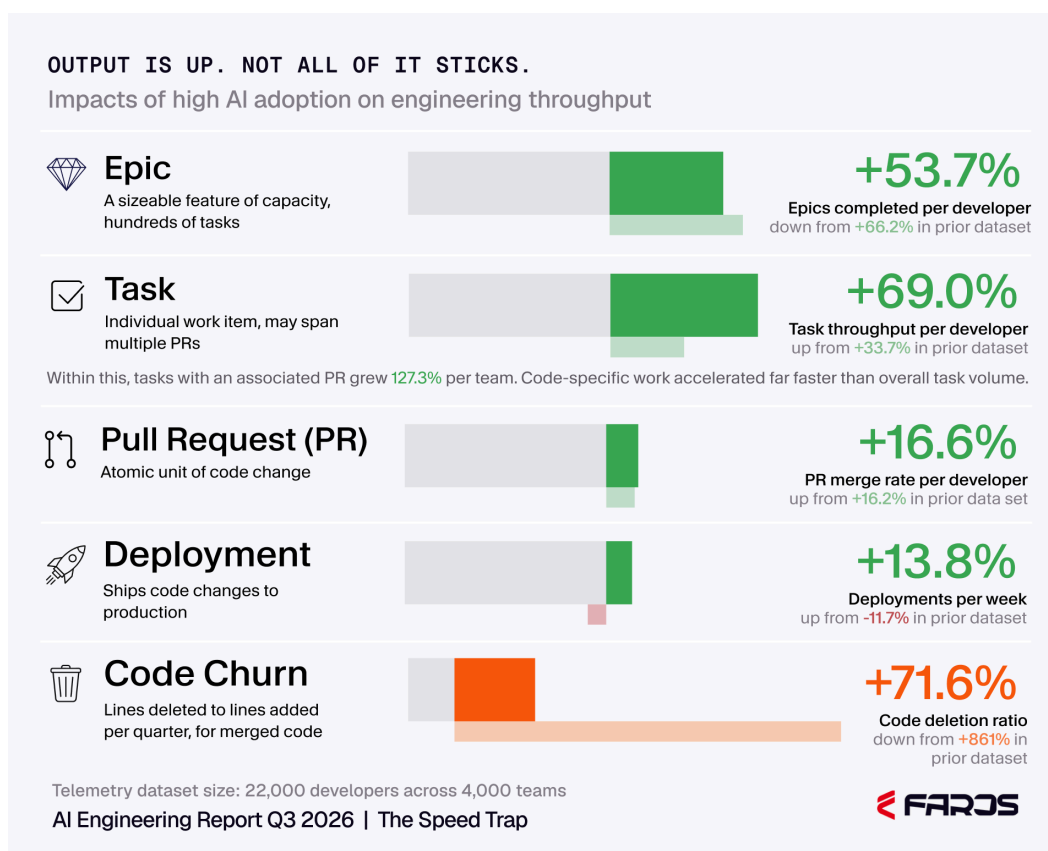
FINDING #3: THROUGHPUT**The pipeline is unblocked, but leaky.**

Output is still up. And this time, it is reaching production. Not all of it works when it gets there.

In April, the throughput section carried an asterisk the size of the finding itself: more code was completing and merging, but deployments per week were falling. The arm that ships was under pressure. That has since reversed.

Throughput metrics:

- **+69.0%** task throughput per developer (up from +33.7% in prior dataset)
- **+53.7%** epics completed per developer (+66.2% in prior dataset)
- **+16.6%** PR merge rate per developer (+16.2% in prior dataset)
- **+127.3%** tasks completed per team with an associated PR (+210% in prior dataset)
- **+13.8%** deployments per week (reversed from -11.7% in prior dataset)
- **+71.6%** code churn (down from +861% in prior dataset)



Deployment frequency is the single most important directional change in this edition. In the prior dataset, high AI adoption teams merged more and shipped less, the clearest evidence that the delivery system was saturating. In this dataset, high AI adoption teams deploy 13.8% more.¹ A higher deployment count is not the same as more working software, however. Pipeline investment, release automation, and process adaptation may have cleared some of the hurdles between merge and production. But this happened while unreviewed merges nearly doubled and QA time exploded, which points to gates having been removed, with the consequences showing up further downstream.

Code churn tells the other half of the absorption story. In the prior dataset, the ratio of lines deleted to lines added increased 861% under high AI adoption. It was the asterisk we placed on every output number. In this dataset, the increase is now only 71.6%. To be exact about what that number is: it is a 71.6% increase in the deletion-to-addition ratio, *not 71.6% of work being redone*. We offered three explanations in April: rework of AI code accepted too quickly, AI-enabled refactoring of legacy systems, and engineers returning to polish code they had already shipped. The collapse of this figure is good evidence that the first explanation likely dominated the transition period. Teams adopting AI for the first time generated enormous volumes of code that was replaced or deleted shortly after being merged. Teams deepening existing adoption generate much less of it. It remains an elevated volume of discarded work with an unresolved cause, however, which calls first for upstream fixes at the authoring stage.

Task throughput is up 69.0% per developer, roughly double the prior dataset's figure. Note that this occurred despite the higher baseline, making it one of the few metrics that grew even though this edition's comparison starts from a higher adoption floor. Tasks completed with an associated PR grew 127% per team, still far faster than general task throughput. AI's most concentrated effect remains exactly where the tools aim it: the act of writing software.

Epic completion, the finding we tied most directly to business outcomes in April, remains strongly positive at 53.7%. Weekly active AI usage also correlates positively with epics completed per developer across the full dataset ($p=0.24$). The correlation is modest, but it is among the strongest adoption-to-outcome correlations we measure. AI is still enabling teams to complete larger bodies of work, not just individual tasks.

¹ Deployment frequency is measured by approximately 10% of the teams in our dataset. The findings are statistically significant within this subset and directionally consistent with the broader data. The reversal from the prior report should be treated as directional; the datasets differ in composition. We flag it prominently because we flagged the decline prominently, and the reversal deserves the same visibility.

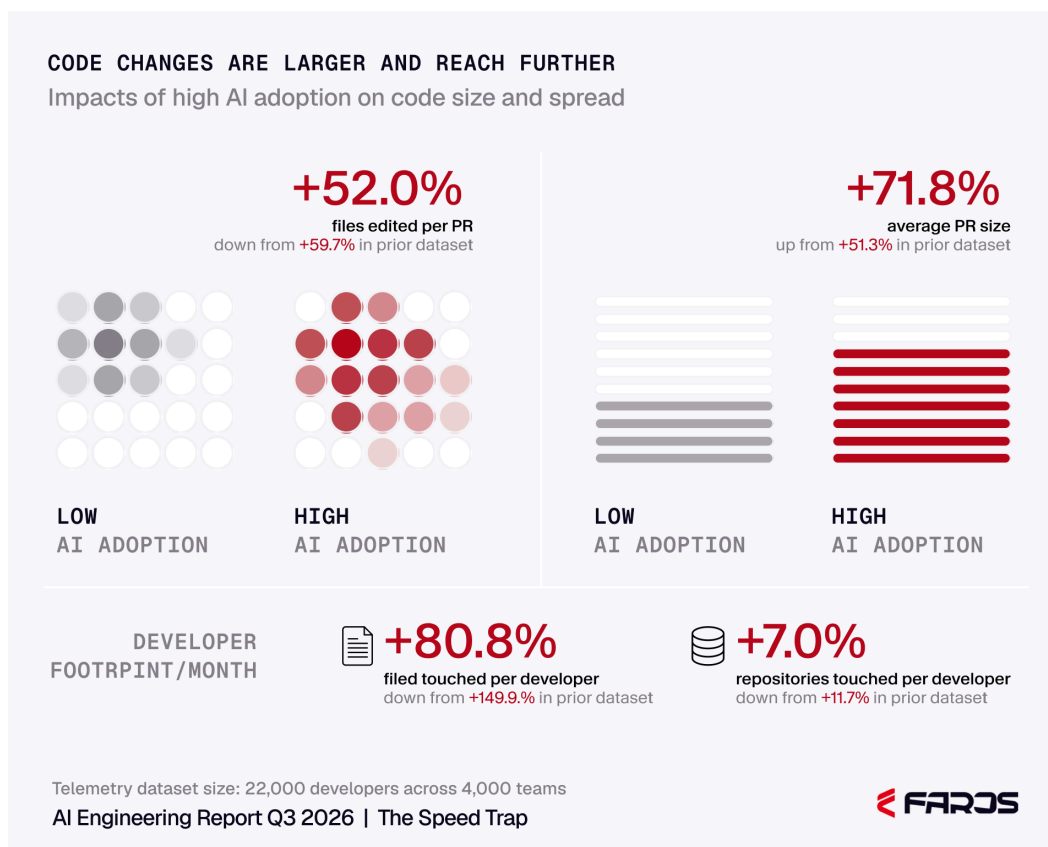
FINDING #4: CODE SIZE AND SPREAD**Changes are growing, but no longer exploding.**

The footprint of a code change keeps expanding. PR size is the one dimension that got worse.

The prior report's findings on the size and spread of changes are confirmed in this dataset, at generally lower magnitudes, with one exception moving the wrong way.

Code size and spread metrics:

- **+71.8%** average PR size (up from +51.3% in prior dataset)
- **+52.0%** average files edited per PR (+59.7% in prior dataset)
- **+80.8%** average files touched per developer per month (+149.9% in prior dataset)
- **+7.0%** average repositories touched per developer per month (+11.7% in prior dataset)



PR size is the outlier. Against a higher baseline that shrinks every other figure in this report, average PR size increased more than in the prior dataset. Changes entering the codebase in a single merge are getting larger in absolute terms and larger relative to everything else we measure. Every consequence we described in April compounds with size, including harder review, wider blast radius, and structural errors that surface-level review often misses.

This matters more in this edition than the last, because of what Finding #5 shows about where those larger PRs are going: straight to merge without being reviewed.

The developer footprint continues to expand, with 80.8% more files touched per developer per month, but at roughly half the prior rate. The pattern across this section matches Finding #3: although the transition shock is fading, the structural shift it revealed is not. AI-assisted changes are permanently larger and permanently broader than the human-paced baseline. The tooling and policy around them have to be built for that reality, starting with size limits enforced at the point of authoring, even before the PR is opened.

Size is a token multiplier. A larger PR costs more to generate, more to review whether the reviewer is human or agentic, and more to regenerate when it comes back. PR size is the one dimension here that worsened against the higher baseline, and it raises the unit cost of every stage downstream of it.

FINDING #5: PRE-MERGE CODE QUALITY**Controls are slipping.**

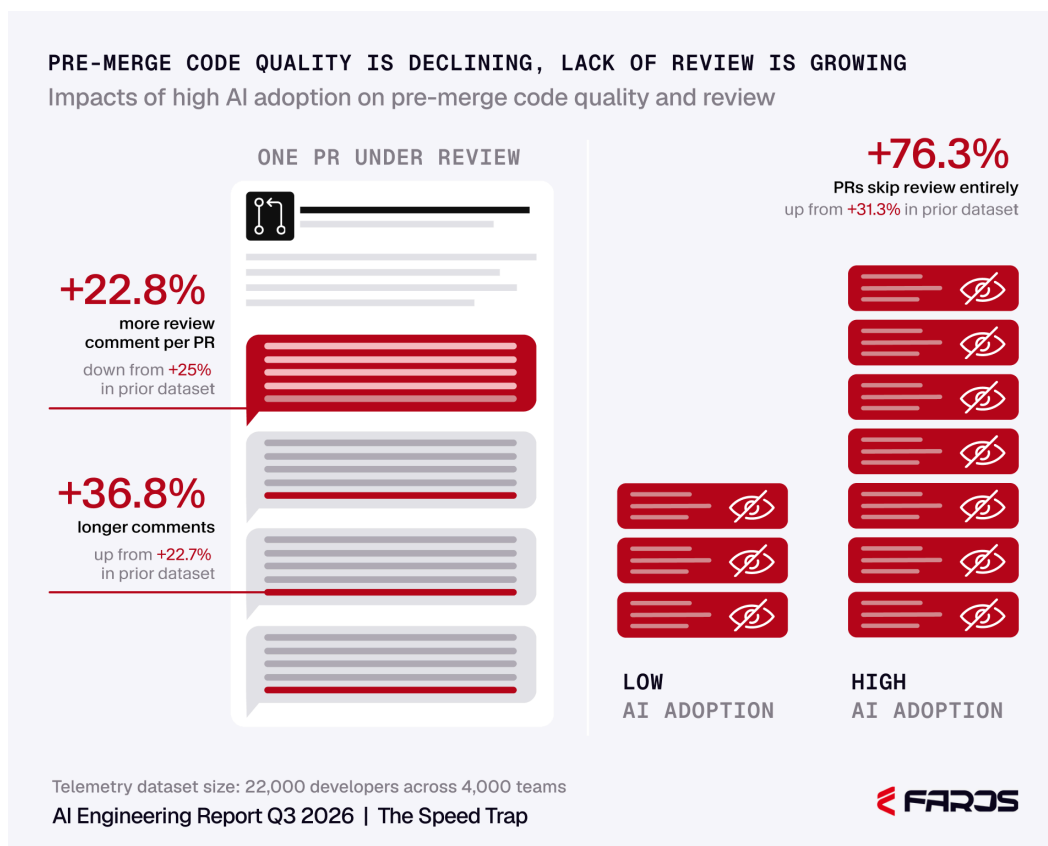
We flagged a 31.3% rise in unreviewed merges as one of the most urgent findings of our last report. It is now 76.3%.

In April, one of the most urgent findings in this report's predecessor was that PRs merged without any review, human or agentic, had increased 31.3%. We wrote that a gate only works if the gatekeeper is not buried under an avalanche.

The avalanche happened.

Pre-merge code quality metrics:

- **+76.3%** PRs merged without any review (up from +31.3% in prior dataset)
- **+22.8%** average review comments per PR (+25% in prior dataset)
- **+36.8%** average length of PR review comments (+22.7% in prior dataset)
- **+22.1%** PRs with merge conflicts
- **-12.0%** share of PR reviews handled by the top 10% of reviewers (-13.2% in prior dataset)



The 76.3% figure carries the most weight in this report, so it needs careful interpretation. Unlike almost every other figure here, it did not shrink as the baseline rose. It is the one governance metric that more than doubled as AI usage deepened. As teams move from moderate to high AI adoption, they are not scaling review to match output. They are routing around it. More code enters the codebase with no eyes on it, human or agentic, than at any point in our data.

This is evidence of a rational response to an impossible constraint. Output has permanently outgrown human review capacity, and in the absence of an enforced policy, individual developers under delivery pressure will make individual decisions to skip the queue. Each decision is locally defensible, but in aggregate it turns into a governance decision no one actually made.

Should all code still be reviewed?

There is a live debate on this question. The case against universal review goes like this: Mandatory review of every change is a convention from an era when humans wrote all the code and review was the primary quality gate. That era is over.

The volume numbers in this report suggest the mandate can no longer be met. Insisting on it produces exactly the queues documented in Finding #6, and a rushed, rubber-stamp approval on an oversized PR is review theater, not meaningful review. If a change is small, low-risk, covered by tests, and gated by CI, static analysis, and an agentic pass, having a human read every line may add more latency than safety. On this view, the rising unreviewed-merge rate is the system rationally shedding a ceremonial gate.

The case for universal review answers that review was never only about defect detection. It is also where knowledge transfers, architectural coherence is maintained, and accountability attaches to a change. That role may become even more important as AI authors more of the code. When no human fully holds the intent behind a change, review may be the last point at which someone is required to understand what is entering the codebase. Taken to the extreme, would anyone want to fly on a plane containing even a small amount of AI-generated code that had never been reviewed by a human?

Our data cannot settle the philosophical question, only the empirical one: when teams in this dataset skip review, are they substituting a different gate, or no gate? We have the answer: no gate. If skipped review reflected a deliberate policy, with tests and CI replacing human review for low-risk changes, we would expect the verification burden to hold steady or shift toward automation. Instead, it is shifting to the most expensive places possible.

Time in QA is up 300.6% (Finding #6), suggesting that checks deferred at review are being performed later, by hand. Monthly incidents are up 125.4%, while mean time to remediate is up 25.8% (Finding #7), meaning more failures are making it downstream and taking longer to resolve. In many cases, that means debugging code that no human ever reviewed in the first place. At the same time, review is

being skipped as PR size rises 71.8% and files per PR rise 52.0%. The pattern is clear: review is being deferred rather than replaced, and that deferral accrues like interest. The later the check happens, the more it costs.

Our conclusion sides with the reformers on the premise, but against the current practice. The question “should humans review every line of code?” now has a defensible answer of no. The mandate has already broken under volume, which is what the 76.3% figure records. The more important question is whether *something* should review every line of code. Our answer remains yes, and for the first time we have evidence about what that something can be. Finding #8 shows that when an agentic gate stands in, first review arrives faster, change failure rate falls, and gate erosion runs at roughly half the general rate.

PRs merged without any review are up 76.3%, more than double the increase we called unsustainable six months ago.

Which gate you choose comes after the more fundamental requirement: an explicit, enforced, auditable rule, plus the instrumentation to know when it is being bypassed. The threshold should also scale with consequence. Nothing here shows that agentic review catches everything, and the more critical the software, the stronger the case for human eyes on every line.

The more urgent debate is about which gate should review which change. What our data condemns is the current default, where review is skipped more because of deadline pressure than policy, one merge at a time, often on the changes least suited to skipping.

The remaining review signals point in the same direction as they did in April, with the same caveats. Comments per PR are up 22.8% and comment length is up 36.8%. Agentic review comments do not distinguish themselves from human ones in this data, and agents comment more verbosely, so some of this inflation is instrumental rather than substantive. Review concentration continues to fall. The top 10% of reviewers handle 12.0% less of the load, nearly identical to the prior dataset, which we read as genuine decentralization of review work, accelerated by agentic review taking the first pass.

Review is becoming more capable and more widely distributed. More reviewers are participating, agentic first passes are handling a growing share of the work, and the load is spreading across the system. At the same time, review is being bypassed altogether at a rising rate.

Finding #8 shows what a more capable review system can achieve when it is applied. This finding reveals the other side of the story: how often no review is happening at all. The gap between those two is the policy decision every engineering organization is currently making, whether intentionally or by default.

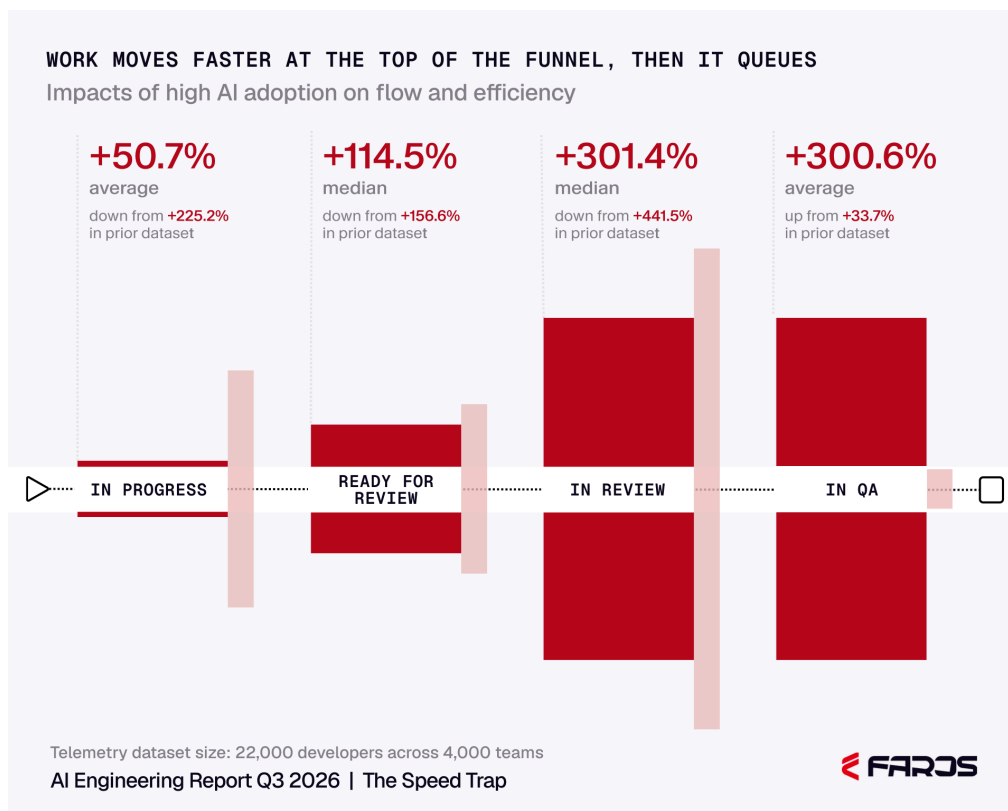
FINDING #6: FLOW AND EFFICIENCY**Work moves faster. Then it queues.**

Active stages recovered substantially. The queues in front of review and QA did not.

In April, every stage of the workflow was slowing down at once. In this dataset, the picture has split in two: the stages where someone is working recovered substantially, while the stages where work waits for judgment got worse.

Flow and efficiency metrics:

- **+50.7%** average time a task spends in progress (down from +225.2% in prior dataset)
- **+114.5%** median time to first PR review (+156.6% in prior dataset)
- **+301.4%** median time in PR review (+441.5% in prior dataset)
- **+300.6%** average time task is in QA (significantly up from +33.7% in prior dataset)
- **+193.0%** average blocking-ticket chain depth
- **+337.8%** lead time from code commit to production (compared to +480.4% in prior dataset)²



² Lead time is based on ~10% of the dataset with high team variance; included as directionally consistent, though caution is warranted. Two other metrics, average Waiting- and In-Review-state time, showed increases too large and errors too wide to include, reported only as directional confirmation that queue-state time remains the dominant flow problem.

Time in progress, the stage where a developer, or increasingly an agent, is actively building, fell from a 225.2% increase to 50.7%. Writing the code is no longer where AI-adopting teams are losing the most time. That is the acceleration working as advertised.

Time in QA went the other way, and dramatically: from a 33.7% increase in the prior dataset to 300.6% in this one. In a dataset where the higher baseline pulls every other flow metric down, QA time increased ninefold relative to its prior reading. The interpretation is uncomfortable but consistent with everything else in this report. As review is skipped (Finding #5) and code arrives in larger, machine-authored units (Finding #4), QA inherits the verification burden that review used to carry. The bottleneck we predicted at review has partially materialized there, and partially moved one stage downstream, to the humans and systems responsible for proving the code actually works.

Review timing itself remains severely elevated even as it moderates: median time to first review is up 114.5%, and median time in review is up 301.4%. Blocking-ticket chain depth, meaning how many tickets sit blocked behind other tickets, is up 193.0%, a signal that the dependencies between fast-moving work items are themselves becoming a queue.

The senior-engineer tax we described in April has not lifted. It has been joined by a QA tax. AI-generated code remains superficially convincing, idiomatic, well-named, and stylistically consistent, with its structural failures beneath the surface. The organizations in this dataset are distributing the cost of finding those failures across review (where it is done), QA (increasingly), and production (Finding #7). The cost itself has not gone away, it moved to wherever the last gate is.

FINDING #7: CODE QUALITY IN PRODUCTION**Each change got safer. The system didn't.**

Each merged PR is far less likely to cause an incident than during the transition period covered in our last report. There are many more PRs, and the incidents that occur take longer to fix.

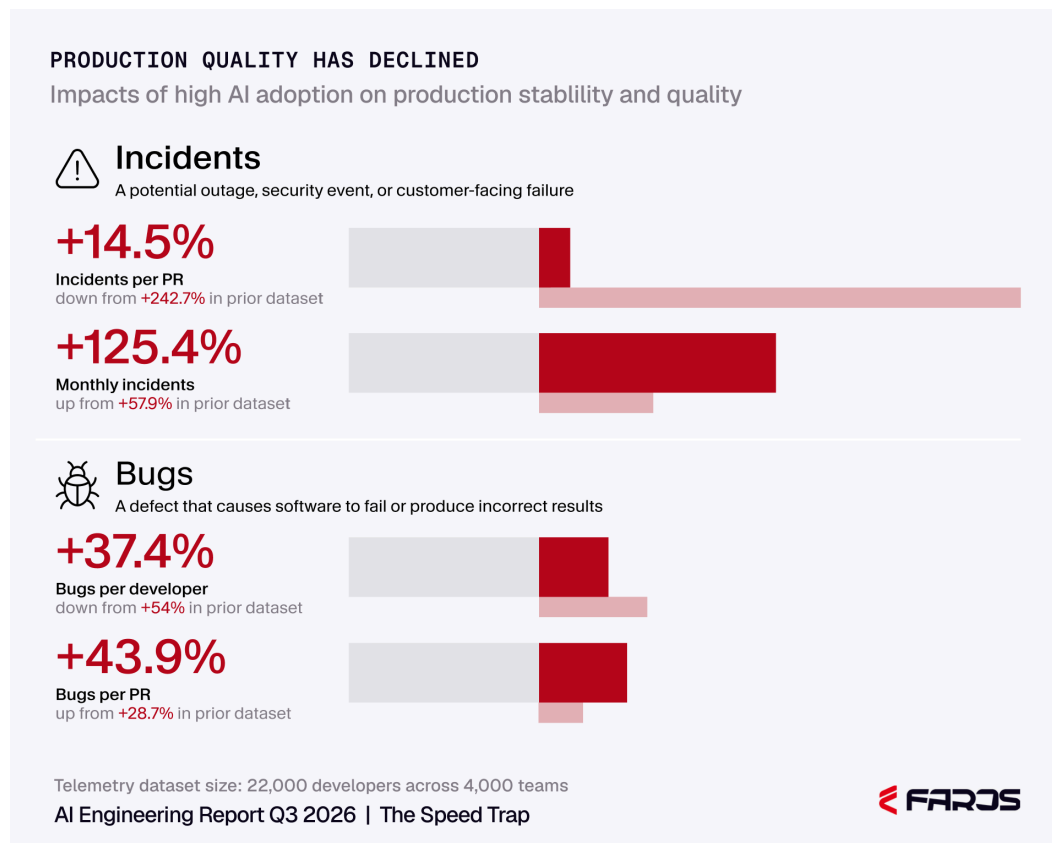
The most cited figure from our April report was the incidents-to-PR ratio: up 242.7%, meaning the probability that a merged change caused a production incident had more than tripled. In this dataset, that ratio is up just 14.5%.

That is a real improvement. This data does not reflect a panic scenario in which per-change quality degrades without limit as AI adoption deepens. Teams deepening AI adoption today ship changes that are only modestly riskier, per change, than their own lower-adoption baseline. The transition-era quality collapse was, in significant part, temporary, reflecting a cost of the adoption ramp rather than a permanent feature of AI-assisted work.

The operational picture is where the strain now lives.

Production code quality metrics:

- **+14.5%** incidents per PR (down from +242.7% in prior dataset)
- **+125.4%** monthly incidents (up from +57.9% in prior dataset)
- **+37.4%** bugs per developer (+54% in prior dataset)
- **+43.9%** bugs-to-PR ratio (up from +28.7% in prior dataset)
- **+25.8%** mean time to remediate (reversed from -3.6% in prior dataset)
- **+42.8%** backlog growth rate month over month (reversed from -29.9% in prior dataset)
- **+6.6%** percentage of reopened Jira tickets (+12.6% in prior dataset)



Reading the first two numbers together is telling. Incidents per PR: up 14.5%. Monthly incidents: up 125.4%. Total production risk is per-change risk multiplied by change volume, and volume won. Each individual merge is far safer than during the transition, yet the aggregate flow of merges produces more than twice the monthly incident load. Organizations tracking only per-change metrics will conclude the problem is solved. Their on-call personnel will disagree.

Two reversals in this table are worth calling out, because both metrics were improving in the prior dataset.

Mean time to remediate flipped from a 3.6% improvement to a 25.8% deterioration. Incidents are not just more numerous; each one is also taking longer to resolve. That slowdown is consistent with the pressure documented in Findings #5 and #6. The same engineers responsible for remediation are also absorbing more of the review and QA burden, while the code behind those incidents is increasingly machine-authored, larger, and in some cases never reviewed. Root-causing a failure in code no human has read is fundamentally slower.

Backlog growth flipped from shrinking 29.9% to growing 42.8%, while average backlog-ticket age rose 42.7% alongside it. In the transition era, AI helped teams burn down accumulated work. In the current deepening era, AI-assisted teams are generating new work faster than they close it, in the form of bugs, follow-ups, and rework tickets.

The bugs-to-PR ratio rose to 43.9%, from 28.7%, even as reopened tickets improved. That combination suggests defects are increasingly being caught and filed before work is declared “done,” which is the better failure mode. But those defects are still creating new work, and they are entering a backlog that is growing once again.

Change failure rate, the share of production deployments that cause a Sev1/Sev2 failure, is new in this dataset and tells a two-sided story that we explore further in Finding #8. The rate remains sharply elevated under high AI adoption, even as the trend is improving. Under high agentic-review adoption specifically, change failure rate falls 24.4%. Where the countermeasure is deployed, the metric responds.³

³ Change failure rate level in this dataset carries a standard error comparable to its estimate and is reported as directional. The trend figure and the agentic-review association are better powered but should still be read as correlational.

FINDING #8: AGENTIC REVIEW

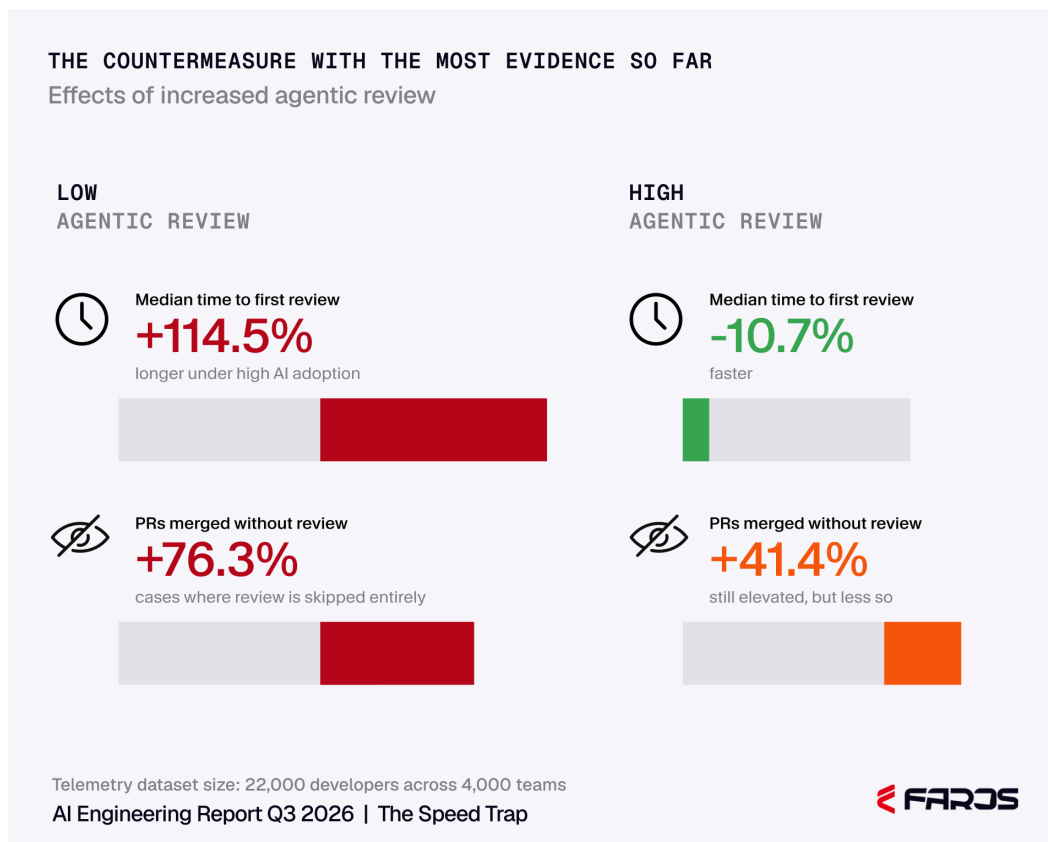
The countermeasure with the most evidence so far.

Where AI review is heavily adopted, first review comes faster, failures come less often, and the load spreads across human and agentic developers. While not a silver bullet, it is the first working mitigation.

Agentic code review is the one intervention that has been adopted broadly enough, and for long enough, to evaluate at scale. In the prior dataset, it had only just crossed 25% of PRs, and teams using it largely performed worse on most timing metrics, which is unsurprising for a capability only weeks into deployment. In this dataset, we compare teams' highest agentic-review quarters to their lowest:

High AI-review adoption metrics:

- **-10.7%** median time to first PR review
- **-24.4%** change failure rate (Sev1/Sev2)
- **-12.9%** share of reviews handled by the top 10% of reviewers
- **-2.7%** build time growth
- **+439.2%** tasks completed with an associated PR
- **+41.4%** PRs merged without review, still up, but at near half the general dataset's rate (+76.3%)



The most important observation was that the time to the first review has fallen. Across the general dataset, the first review takes 114.5% longer under high AI adoption. However, on teams where the first reviewer is an agent, it comes 10.7% faster. The queue in front of review, the place where we said work sits queued up with no one working on it, is exactly what an always-available agentic reviewer eliminates. Change failure rate falling 24.4% on the same teams suggests the agentic first pass is catching a meaningful share of what would otherwise ship broken, before a human reviewer engages or indeed in place of one.

The decentralization signal completes the picture. The top-10% reviewer share falls further on high-agentic-review teams than anywhere else in the dataset. The senior engineers we described in April, who were consumed by unraveling plausible-sounding code, are the direct beneficiaries. An agent that handles structure, formatting, and surface-level correctness before a human looks at the code frees the senior reviewer to do the one thing agents still cannot: reconstruct intent and judge whether the code solves the right problem.

Three caveats:

First, this is correlational. Teams that deployed agentic review early and heavily may simply be the more disciplined engineering organizations, and discipline improves these metrics with or without agents.

Second, agentic review inflates its own paper trail. The strongest raw correlations in our dataset are between agentic-review adoption and review-comment length ($p=0.40$) and review-comment count ($p=0.30$). Comment volume tells you that agents are commenting, and nothing about review quality. The timing and failure-rate results above are the meaningful signals, because they are not subject to that inflation.

Third, agentic review does not stop gate erosion, but it does slow it. Unreviewed merges still rise 41.4% on teams with high agentic-review adoption, compared with 76.3% overall. That is consistent with agents absorbing review volume that might otherwise bypass the gate entirely. Agentic review slows the erosion, but does not eliminate it.

Our April conclusion stands: the fix is at authoring, not review. Review, whether human or agentic, exists to catch mistakes, and the goal remains to reduce the number of mistakes that reach it. Agentic review is measurable today because it was deployed early and at scale. When used as an enforced gate rather than an optional pass, it can slow the bleeding while organizations rebuild authoring-stage quality. It buys time, but it is not a silver bullet.

WHAT ENGINEERING ORGANIZATIONS SHOULD DO

The controls must be rebuilt as policy. Here is what that requires.

Engineering organizations are getting better at absorbing AI-scale output, but some of that progress is coming at the expense of existing controls: review is being skipped at twice the prior rate, remediation is slowing, and operational load is compounding.

Agentic authoring is still below 5% of PRs for most organizations, but the vanguard, now at 13–14%, is a preview of what's coming. Every eroded gate will face far more pressure once the author is autonomous. The window to rebuild those gates – through enforced policy, agentic controls, and less reliance on heroic human effort – is closing.

One discipline runs through the recommendations below: **Token Engineering**.

Token Engineering is the closed loop of observing, optimizing, and governing how an organization spends on AI. In the agent era, spend is increasingly metered by token and grows with every retry, oversized change, and unit of work that has to be redone. The findings in this report show where that consumption can become inefficient or produce downstream cost.

- **Observing** means attributing token consumption to teams, tools, models, and tasks, and measuring it against outcomes: spend per merged PR, per completed task, per incident avoided, etc. It also means making visible the engineering signals that expose where paid work is being duplicated, including restarts, churn, time in QA, and unreviewed merges.
- **Optimizing** means acting on what you observe so that fewer tokens produce more shipped, working software. That includes provisioning better context, routing work to the right models, and constraining scope. The largest lever is **context engineering**: deliberately providing agents with the organizational knowledge they need to author code effectively, including codebase standards, architectural intent, system history, and testing requirements.
- **Governing** means turning the result into enforceable policy that holds as volume grows. That includes deciding which gate reviews which change, how much scope an agent may take on in a single PR, and how much a team, tool, or task is allowed to spend. Spend governance is already spreading. Change governance is what has eroded, and it is the area most organizations need to rebuild first.

The loop closes on a simple observation: every restart, every churned line, every QA rejection is work an organization paid for twice, once in tokens and once in the human time to catch and redo it.

The recommendations below are grouped by the three legs of that loop.

What comes next: our recommendations

The data in this report is diagnostic. The ten recommendations below translate it into actions you can take: **Observe (#1–#4), Optimize (#5–#8), and Govern (#9–#10)**. Start with the ones you can measure today.

#	Action	First step this week	Metric to watch
1	Track unreviewed merges as a first-class risk metric	Pull last month's unreviewed-merge rate	% of PRs merged with zero review
2	Instrument QA as the next wall	Split the tasks sitting in QA by PR origin	Tasks and time in QA, by origin
3	Watch and price work restarts	Compute tokens spent on restarted tasks	Restart rate; restart cost
4	Measure AI spend per outcome, not per seat	Compute spend per merged PR for one team	Spend per outcome
5	Build a context engine	Baseline restarts, QA time, and rework churn	Those three numbers, trending down
6	Feed human review comments into agent context	Route one team's review findings into agent rules	Repeat-issue rate in reviews
7	Enforce PR size limits before the PR opens	Correlate PR size with incidents in your data	Average PR size; failure rate by size
8	Run the churn loop, denominated in dollars	Re-baseline churn alerts to post-transition levels	Cost of deleted recent lines
9	Define which gate reviews which change, as enforced policy	List repos with no defined review requirement.	% of repos with defined policy
10	Deploy agentic review as infrastructure	Enable it on every PR for one team	Time to first review; change failure rate

OBSERVE

1. Track your unreviewed-merge rate as a first-class risk metric.

Alongside **incident-to-PR ratio and monthly incidents normalized by change volume**, add the percentage of PRs merging with zero review.

These three numbers together tell you whether total production risk is rising and whether your controls are keeping pace with volume. If unreviewed merges have more than doubled since you began scaling AI adoption, you are inside the pattern this report documents.

Pair the measure with bug severity data to see the full picture across severity levels.

2. Treat QA capacity as the next wall, and instrument it now.

Split the tasks sitting in QA by the origin of their PR: **human-authored, AI-assisted, or agent-authored**, and by PR size.

Then identify which verification work is repeatedly occurring in QA and push what can be defined mechanically upstream into the agent's definition of done. That includes static analysis thresholds, linting rules, test coverage minimums, and the success criteria defined in tests.

The goal is for the agent's definition of done to mean **passing the specification rather than merely producing plausible code**.

This recommendation follows directly from the report's finding that time in QA is up 300.6%, the largest flow deterioration in this dataset, and that QA is inheriting verification burden as review is skipped.

3. Watch work restarts as your primary context-quality signal, and price them.

Track tasks that return to in-progress after moving to another stage, and connect those restarts to AI consumption.

For organizations that can attribute AI consumption to tasks, compute the **restart bill directly: the tokens consumed by work that returned to in-progress**. A restarted task represents tokens spent on an output that progressed far enough to move forward and then had to be revisited, plus the tokens required for the next attempt.

If restart rate is climbing alongside AI adoption, examine what context your agents operate with.

The fix remains upstream: **better context provisioning before the agent begins, rather than manual correction after the restart.**

4. Measure AI spend per outcome, not per seat.

Attribute AI consumption to **teams, tools, models, and tasks**, and measure spend against outcomes such as **merged PRs or completed tasks**, rather than seats.

Then classify consumption into productive work, recoverable inefficiency, and waste.

The waste categories are already visible in this report's findings. They include restarted work, churned code, and sessions that run away without converging. What is missing in most organizations is the denominator that lets those categories be expressed as AI spend.

The objective is to move AI measurement from access and utilization toward the economics of what the consumption actually produces.

OPTIMIZE

5. Build a context engine for your engineering environment.

Provide agents with the organizational knowledge they cannot derive from the current state of the codebase alone.

That includes the **history of changes, the reasoning behind them, accumulated architectural decisions, codebase standards, security constraints, and testing requirements.**

Feed that context into agents before authoring begins.

Then use **restarts, QA time, and rework churn** as the KPIs of the context engine. These are the signals this report identifies as places where missing intent or inadequate authoring-stage context can surface downstream.

Close the loop by feeding human review findings and churn investigations back into the context available to future work.

6. Treat every human review comment as context the agent was missing.

Analyze human and agent review comments separately.

Agent comments address structure, formatting, and surface-level patterns. Those findings are valuable, but many can ultimately be caught earlier in the workflow.

Human review comments often contain a different type of information: **organizational knowledge, conventions, constraints, or architectural intent that existed in a senior engineer's head and not in the agent's context.**

When a human reviewer identifies something that should change, capture the finding and feed it back into the agent's context as a generalized rule that can prevent the same issue in future PRs.

Track the **repeat-issue rate in reviews** to determine whether that knowledge is actually being absorbed upstream.

The goal is to avoid paying repeatedly for the same human intervention.

7. Set PR size limits for your coding agents and enforce them before the PR opens.

Build scope constraints into agent instructions and enforce hard limits at the tooling level.

Best practice for agentic development follows a **research, plan, implement** sequence. The plan phase should produce a scope that ships as a distinct, self-contained unit able to pass the full pipeline independently.

Use your own data to determine where those constraints should sit. Correlate PR size with incidents and bugs, and track average PR size and failure rate by size.

If incident and bug rates correlate with PR size in your environment, you have the case for making the limit blocking.

8. Keep the churn investigation loop, recalibrate its thresholds, and denominate it in dollars.

Re-baseline churn alerts to the post-transition environment.

Against a calmer baseline, investigate spikes in deleted-recent-lines and use **Git-level line provenance** to distinguish rework of AI output from deliberate refactoring.

Then add the token dimension.

Churned recent lines represent output that was generated, reviewed or QA'd, merged, and then deleted, with every stage paid for. Denominating the signal in dollars rather than lines turns a code-hygiene metric into a **rework tax** that finance can understand.

Use that cost to inform upstream investment in restart reduction, context engineering, and the feedback loop from human review.

GOVERN

9. Decide, as policy, which gate reviews which change. Then enforce it.

Define explicitly:

- which areas of the codebase require **human review**;
- which may be gated by a **trusted AI agent backed by tests and CI**; and
- which require **both**.

An **agent-only gate** should have a precise meaning: an AI reviewer's pass plus green CI, static analysis, and coverage thresholds, with no human approval, permitted only on codebase areas the organization has explicitly designated.

Slice the policy by risk. High-stakes systems, security-sensitive paths, and customer-facing services warrant human eyes. Lower-risk, well-tested areas may be appropriate for agent-only review.

The objective is to replace review decisions made one merge at a time under delivery pressure with an **explicit, enforced, auditable rule** governing which gate is appropriate for which change.

10. Deploy agentic review as infrastructure, not as an experiment.

Run agentic review on every PR by default, tune it against your codebase standards, and route its findings into the authoring harness so the same issue stops recurring.

The policy in Recommendation #9 still determines where human approval is required. **Agent approval should not be treated as equivalent to human approval in high-risk paths.**

Instrument the review agent's precision. An agent that comments verbosely on everything trains developers to ignore it, recreating the unreviewed-merge problem behind a green checkmark.

Also instrument the economics before scaling the approach. This report shows measurable associations between high agentic-review adoption and improved review timing and change failure rate, but **no one, this report included, has yet shown what the review-heavy approach costs per working shipped outcome.**

Measure that alongside the operational benefits.

And our recommendation on head count stays the same: repurpose engineering capacity, do not reduce it. The temptation is stronger now than in April because the throughput numbers are better and the per-change quality numbers no longer look like a crisis. Resist it on the operational evidence. Incidents are up 125% monthly, remediation is 25.8% slower, and backlogs are growing again. The engineers you would cut are the ones absorbing the review, QA, and incident load that AI volume creates.

The right move remains repurposing: redirect capacity toward the gates, the token instrumentation, and the context engine described above, all the scaffolding that makes AI adoption sustainable. Stabilize the process, demonstrate with these signals that it is under control, and then make workforce decisions from a position of confidence.

These recommendations assume a baseline of visibility into your engineering environment. If you cannot answer most of these questions today, the visibility gap is the first problem to solve. You cannot govern what you cannot see. And this edition adds a corollary: you cannot un-skip a review you never knew was skipped, and you cannot recover tokens you never knew you spent.

APPENDIX

Research methodology

This study analyzes the relationship between AI adoption and engineering outcomes across a large sample of development teams, drawing on the most recent 12 months of telemetry data. Telemetry was aggregated from task management systems, IDEs, static code analysis tools, CI/CD pipelines, version control systems, incident management systems, and HR system metadata.

The analysis focuses on development teams as they increased AI tool adoption above the 50% weekly active user threshold. AI adoption includes developer-facing coding assistants, such as Claude Code, Cursor, GitHub Copilot, Windsurf, and similar tools, as well as autonomous agents integrated directly into the SDLC.

How we isolated the signal:

- Standardized all metrics per company to remove inter-organizational variance.
- Used Spearman's rank correlation (ρ) to assess relationships between metrics and AI usage.
- Reported only metrics with data from at least 6 companies and statistically significant correlations (p -value < 0.05).
- For each team, we calculated the percent change in metric values between the two quarters of lowest AI adoption and the two quarters of highest AI adoption within the observation period.
- Excluded outlier data and metrics with insufficient historical coverage.

This approach enables comparisons within each company over time and avoids misleading aggregate assumptions across different organizational structures.

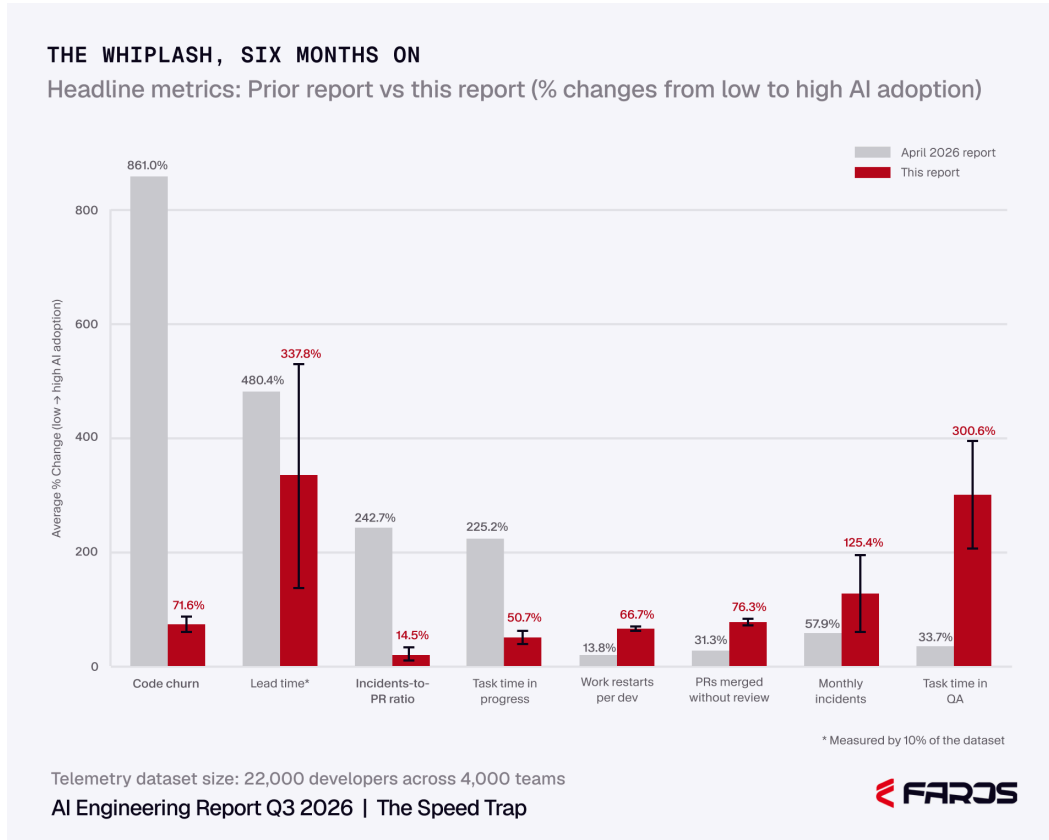
Reading this edition against the prior report: Three metrics in this dataset carry standard errors comparable in magnitude to their estimates: average time in the Waiting state, average time in the In-Review state, and change failure rate level. We report them as directional signals and have excluded them from headline claims. Where we compare a figure to the prior report, the prior figure is quoted as published in April 2026, and the comparison should be read in light of the deepening-vs-transition distinction described above.

Note on what this telemetry does not include: none of the sources above records AI consumption. Token spend will be included in our next report.

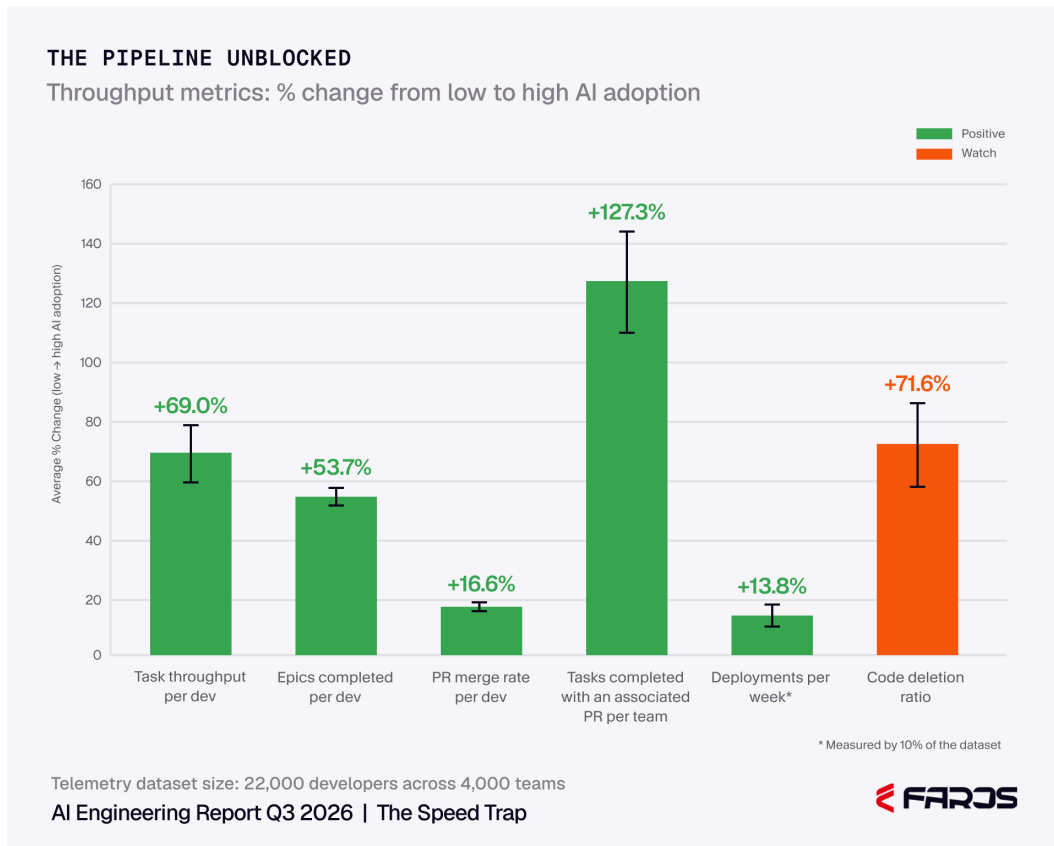
Versioning note: This version of the report reflects analysis as of August 2026.

Supporting Charts

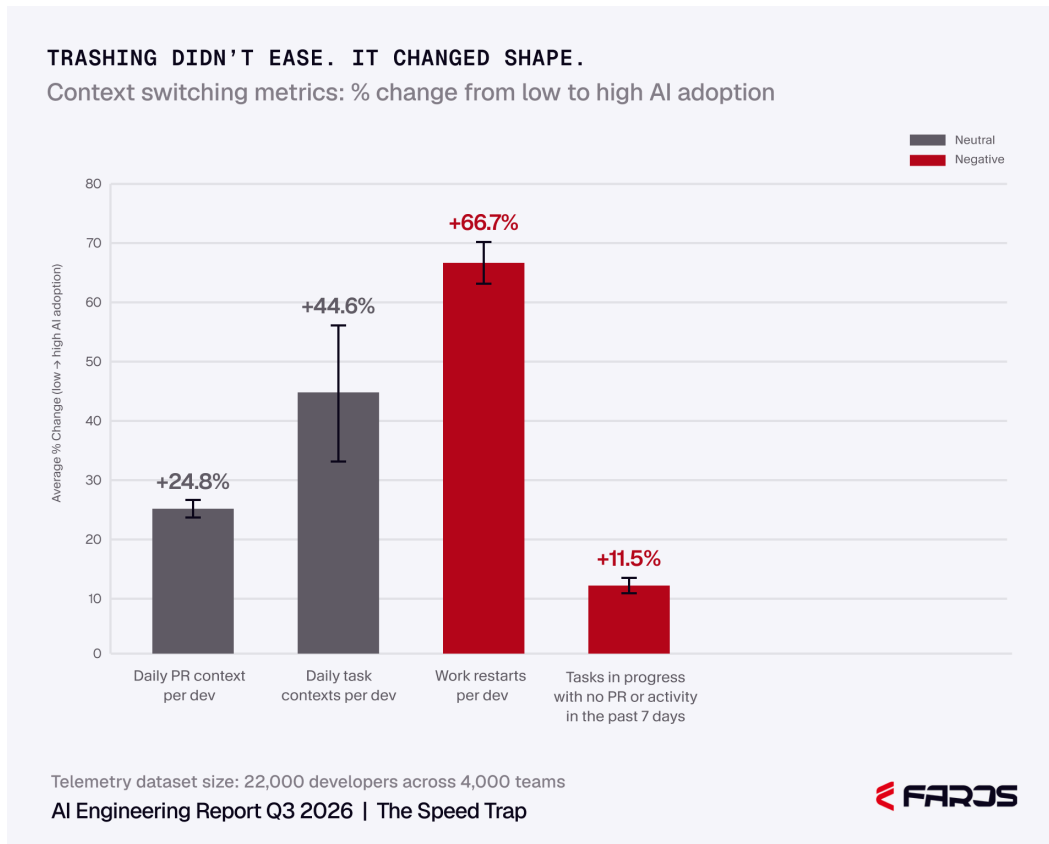
The whiplash, six months on:



Throughput:



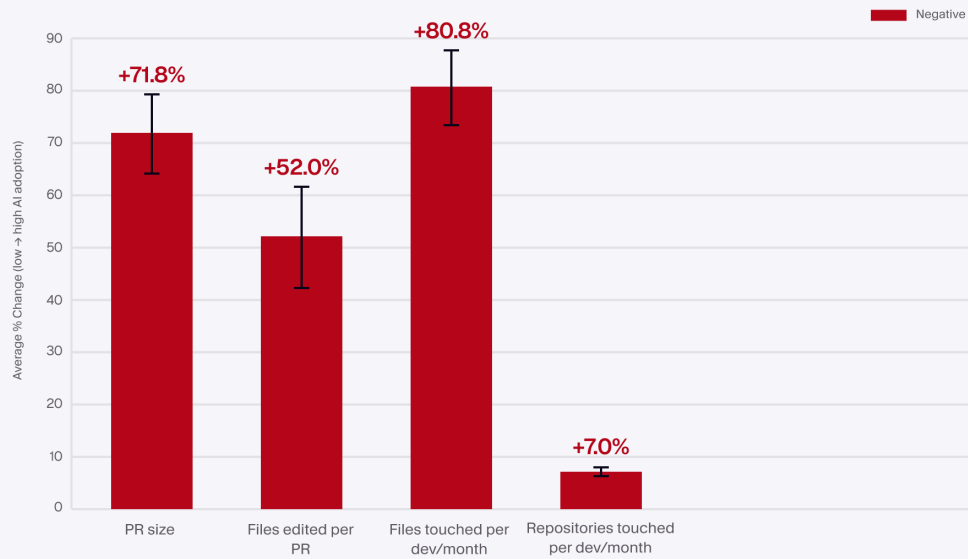
Developer cognitive load:



Code complexity:

CHANGES ARE STILL GROWING. NO LONGER EXPLODING.

Code complexity metrics: % change from low to high AI adoption

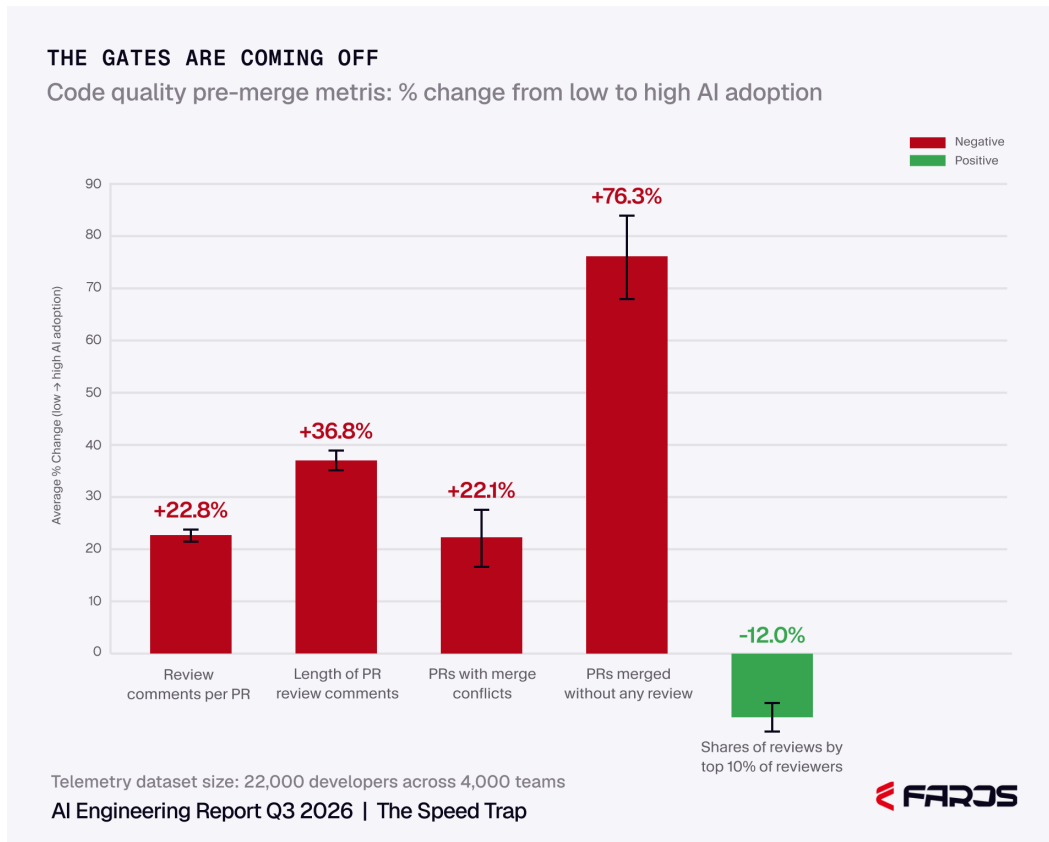


Telemetry dataset size: 22,000 developers across 4,000 teams

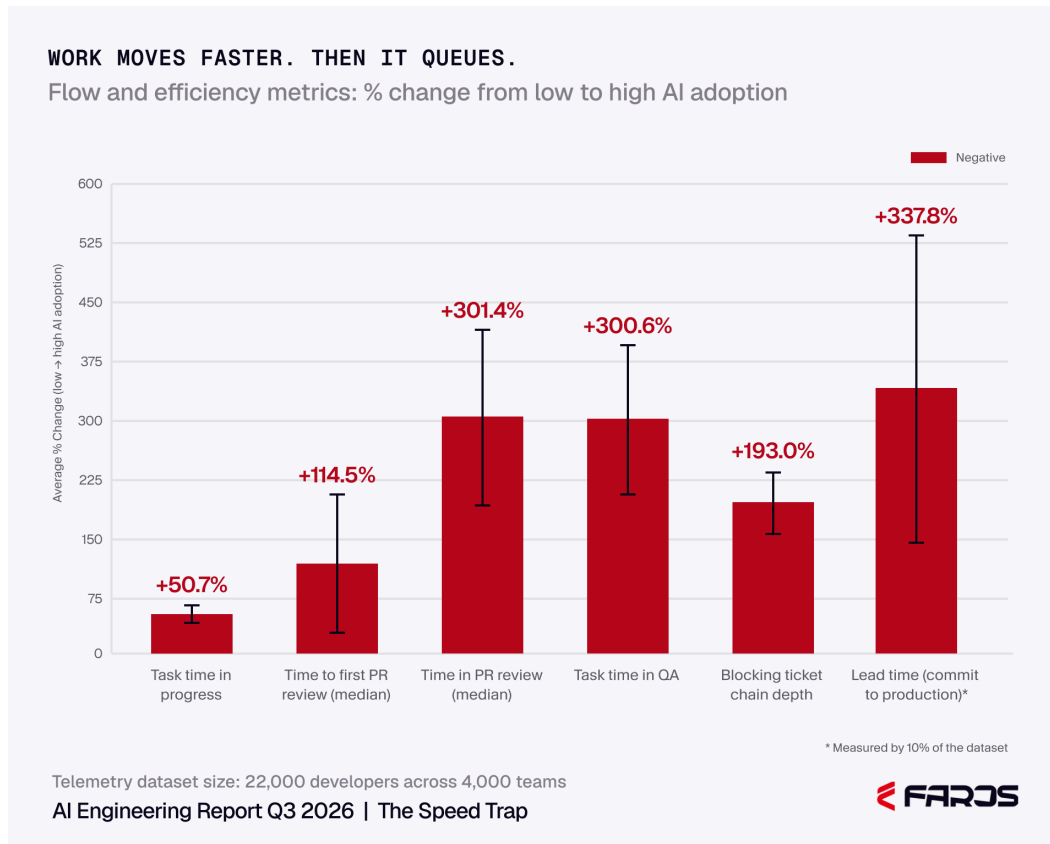
AI Engineering Report Q3 2026 | The Speed Trap



Pre-merge code quality:



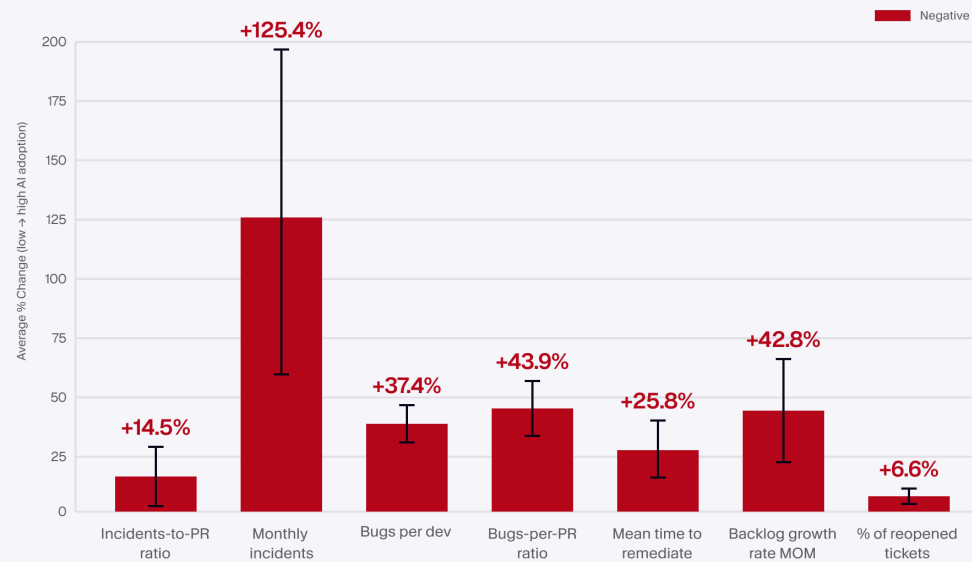
Flow and efficiency:



Production code quality:

PER-CHANGE RISK STABILIZED. OPERATIONAL STRAIN DIDN'T.

Production code quality metrics: % change from low to high AI adoption



Telemetry dataset size: 22,000 developers across 4,000 teams

AI Engineering Report Q3 2026 | The Speed Trap



Agentic review:

