CodeRabbit

The LeadDev

AI Code Review Report 2026

How 377 engineers and engineering leaders are trusting code they didn't write, and what keeps them busy as AI output grows.



Introduction

Your team ships more code than ever, and review capacity hasn't grown to match the increased volume of agentic code. That gap in code written vs. code reviewed runs through every part of LeadDev's AI Code Review Report 2026.

We surveyed 377 engineers and engineering leaders about how AI has changed the way their organizations write, review, and ship code. The answers line up with uncomfortable consistency. AI coding tools are everywhere. Pull requests (PRs) have multiplied and grown. The human-review process meant to catch problems before production is buckling under the load.

AI coding has changed how we write code, and the rest of the software development lifecycle is starting to change as well. Most teams using AI review tools report better outcomes in terms of code quality, and plenty of organizations are getting real value. The gains are uneven, though. Governance is thin, measurement is patchy, and most organizations can't yet say whether their investment in AI review is paying off.

The report is split into five sections: how far AI adoption has gone, what it has done to review workloads, how humans review AI code, how organizations govern it, and what AI review tools are delivering. The conclusion turns the data into practical moves for engineering leaders, drawn from the organizations where review is holding up.

Michael Hill
Managing Editor, LeadDev

LeadDev

A note from CodeRabbit

As agents take on more implementation tasks, engineering teams are moving away from reading the code they are responsible for shipping. They now need to govern how agents make decisions and verify the resulting changes against business intent, architectural constraints, and company standards. Engineering judgment remains central, with responsibility extending to how software behaves, stays secure, and evolves in production.

CodeRabbit helps engineering organizations scale agentic development with confidence that the software they ship serves its intended purpose, meets their standards, and can be securely maintained as it evolves. CodeRabbit is the governance and control layer for agentic software development, enabling teams to trust the code they ship while moving at agentic speed.

CodeRabbit

Key findings

45% say review capacity is the biggest drag on delivery

Half see **25%+** more PRs per developer; **67%** say PRs got bigger

38% say reviews got slower

26% fully understand the AI code they review

68% don't always know which changes need deeper scrutiny

47% of developers rely on AI coding agents to review their work

43% say it's too early to tell ROI

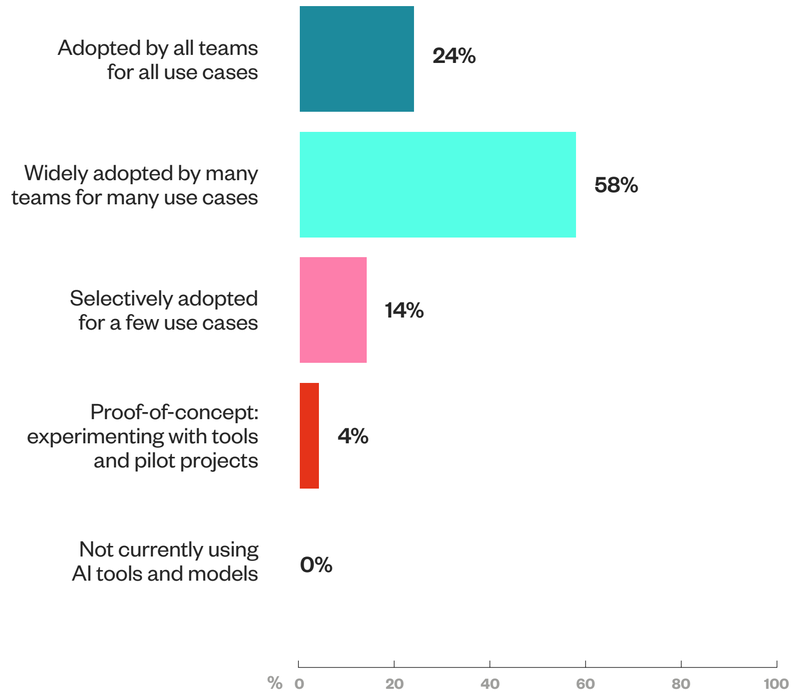
AI coding tools are now standard issue



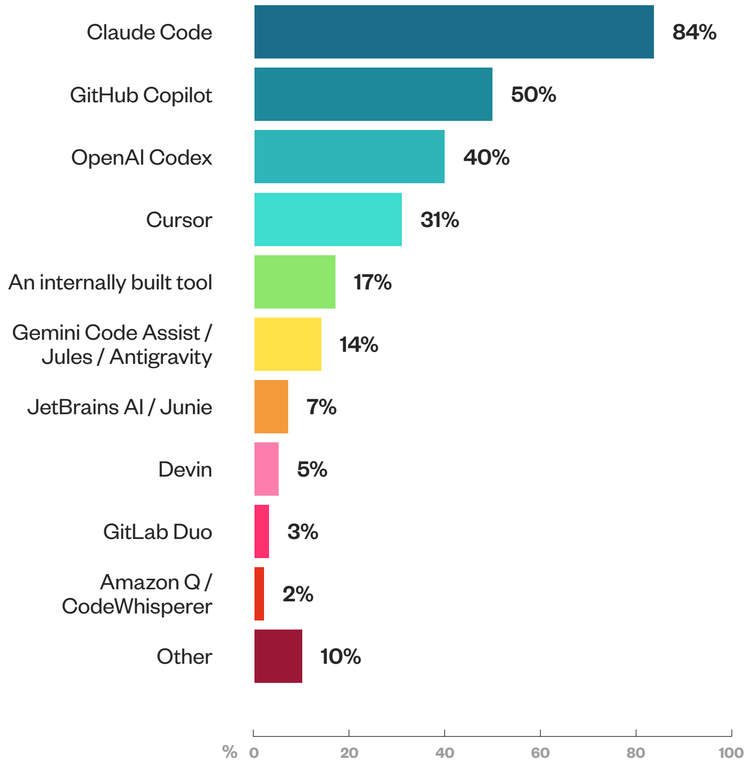
The AI adoption debate is over: 95% of engineering organizations we surveyed have already adopted AI-powered developer tools, including 24% where every team uses them for all of the use cases. Just 14% have only selectively adopted these tools for a few use cases, while 4% are still running pilots.

Claude Code leads by a wide margin with 84% of organizations using it as a coding agent or assistant. GitHub Copilot (50%), OpenAI Codex (40%), and Cursor (31%) are other tools that have attained significant user bases. Only 17% use an internally built AI coding tool or assistant, and 14% use Google's Gemini Code Assist, Jules, or Antigravity. Smaller shares use JetBrains AI or Junie (7%), Devin (5%), GitLab Duo (3%), and Amazon Q (2%).

Which of the following best describes the stage of your engineering organization's adoption of AI-powered developer tools?



Which AI coding agents or assistants does your organization currently use?



Add those numbers up and the average organization runs around two and a half AI coding tools at once. That matters for review, as each tool has its own habits, failure modes, and ideas about sensible PRs. Your reviewers have to absorb all of them.

The tool choice also shows up in how teams experience review:

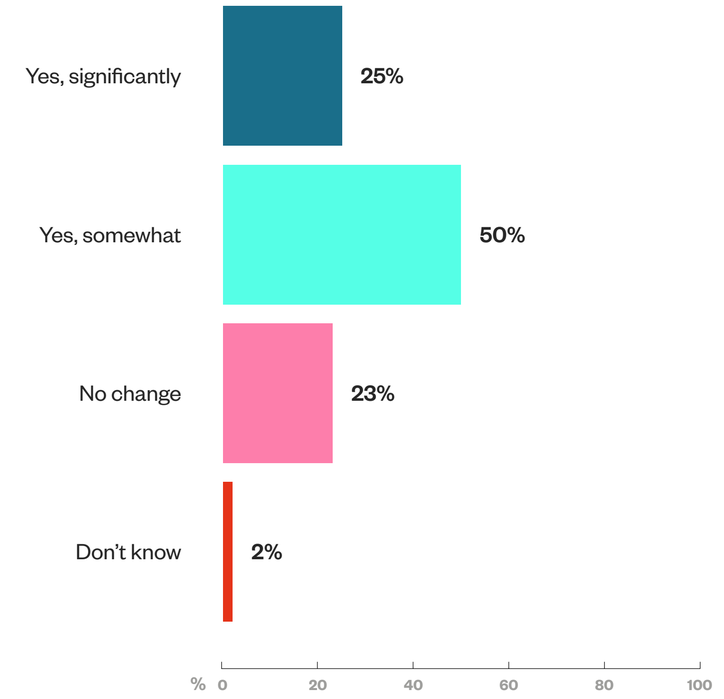
- Cursor users are more likely than average to call code review capacity their biggest bottleneck (56% vs. 45%).
- Codex users are more likely to rely on the AI agent to review its own work (59% vs. 47%).

The population writing code is changing too, with 75% saying that the range of people and systems that write code at their organization has expanded in the past 12 months; 25% say it has expanded significantly. Wider authorship means more code from contributors with less history in your codebase, and all of it ends up in the same review queue.

What does this mean?

With AI adoption close to universal, the release bottleneck has shifted to review. Your review process now has to handle more code, from more authors, shaped by more tools than it was built for.

Has the scope of who can write code, in terms of both people and automated systems, expanded at your organization in the last 12 months?




Review has become the
bottleneck in delivery

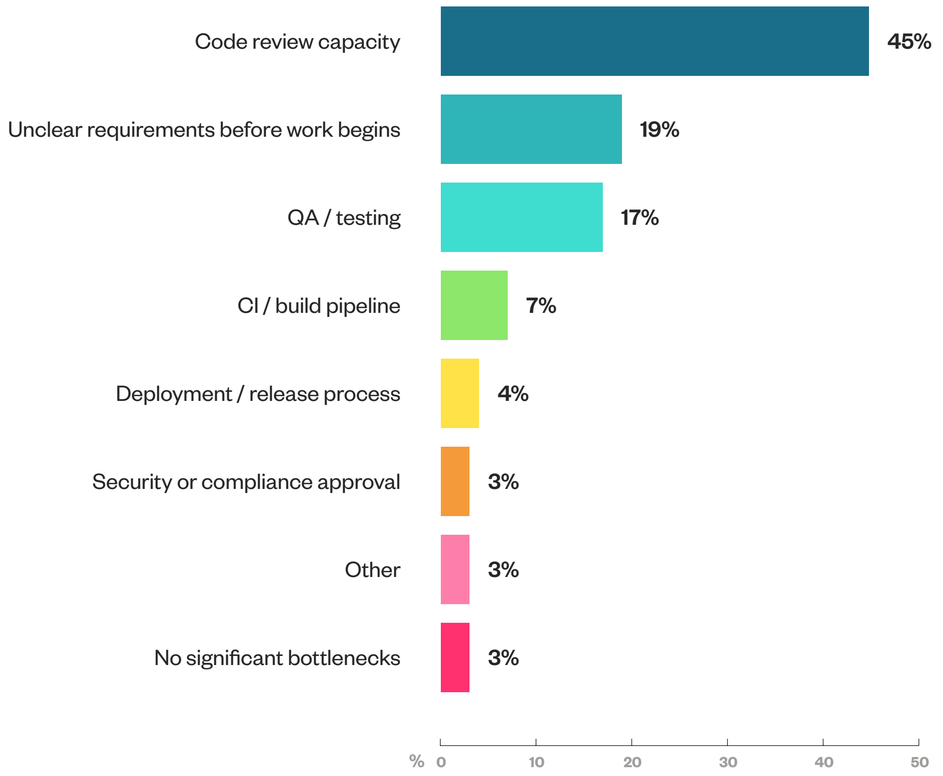
If you want one number to show your leadership team, use this one: 45% of respondents say code review capacity has become the biggest bottleneck in their software delivery process since AI adoption. Unclear requirements came second at 19%, then QA and testing (17%), CI and build pipelines (7%), deployment processes (4%), and security or compliance approval (3%). Only 3% report no significant bottleneck at all.

Half of respondents say the average number of PRs per developer has risen by at least 25% since adopting AI coding tools, while 27% report an increase of 50% or more and one in ten has seen it at least double.

They're also getting bigger: 67% say the typical PR has grown in lines of code, and 32% say it has grown significantly. Just 5% say PRs have shrunk.

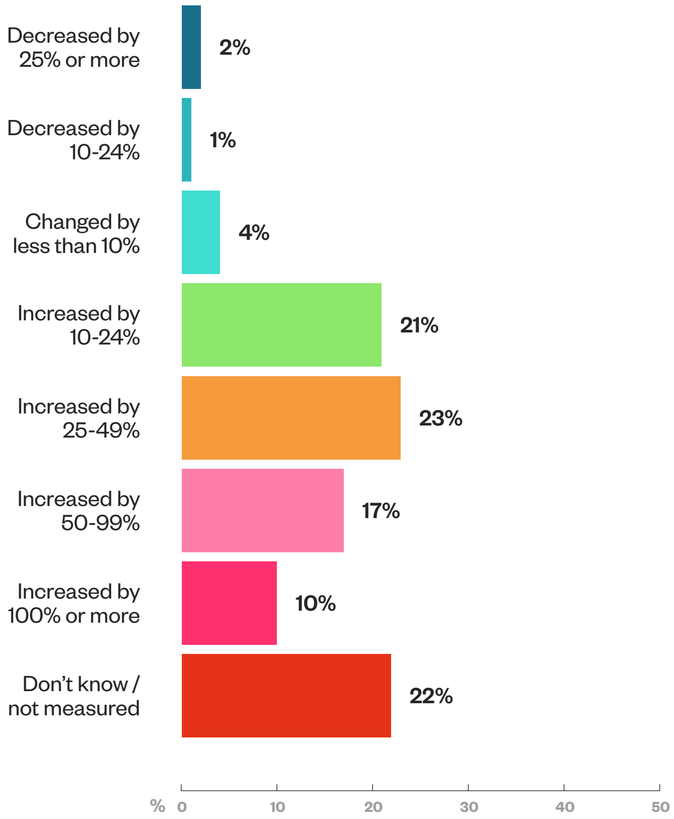
Review time has followed a similar pattern, with 40% of respondents stating the time a typical developer spends reviewing PRs is up at least 25% (19% say it is up 50% or more).

Where is the biggest bottleneck in your delivery process since AI adoption?

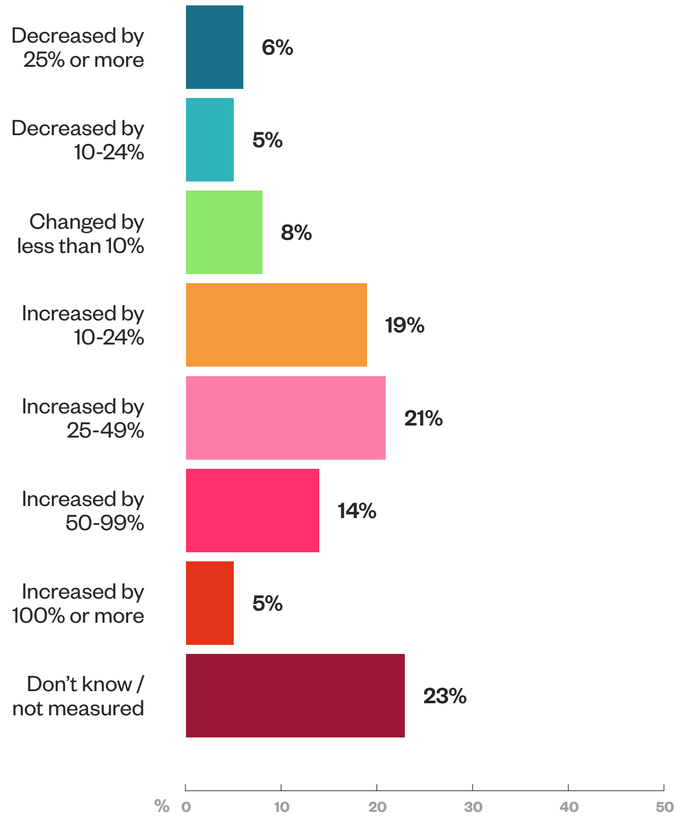


As more code arrives in larger chunks, reviewers spend more time on it, and the cycle slows down. More than a third (38%) say review cycles have gotten slower since AI adoption, against 25% who say they've gotten faster.

Compared with before your organization adopted AI coding tools, how has the average number of pull requests opened per developer changed?



Compared with before your organization adopted AI coding tools, how has the amount of time a typical developer spends reviewing pull requests changed?



The numbers move together:

- Where every team uses AI for all use cases, 46% report a 50%+ jump in PRs per developer, compared with 27% overall.
- Organizations with a 50%+ jump in PRs are more than twice as likely to say review time has risen by 50% or more (41% vs. 19%).
- Among teams whose PRs grew significantly larger, 55% say review cycles slowed, compared with 38% overall.
- Teams with slower cycles are more likely to name review capacity as their biggest bottleneck (56% vs. 45%).

Companies with fewer than 50 employees are more likely to report a 50%+ rise in PRs (45% vs. 27%), and also more likely to see fewer escaped defects in AI-assisted code (31% vs. 19%). The group is small, at around 57 respondents, but one explanation fits: people at smaller companies tend to share more context, which makes extra volume easier to absorb.

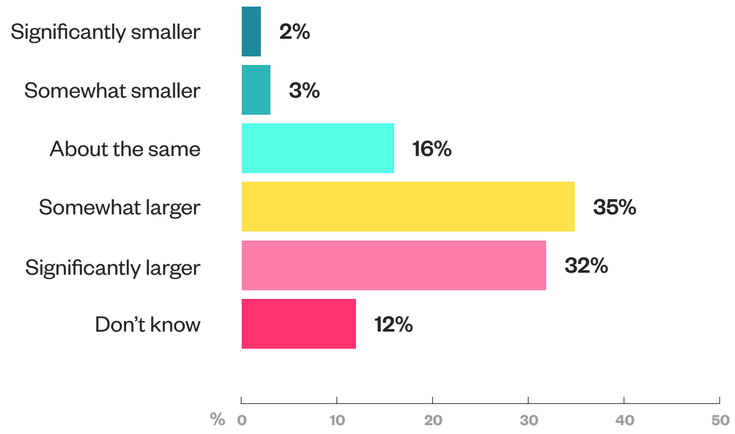
One more detail is worth a pause. **Teams whose reviews got faster are more likely to say QA and testing is now their biggest bottleneck (26% vs. 17%)**. Speed up one step in the SDLC and the queue moves to the next one. Fixing code review alone won't fix software delivery.

Finally, 22% of teams don't know how PR volume has changed, and 23% don't know how review time has changed. Close to a quarter of organizations are making decisions about AI-driven workloads without the numbers in front of them.

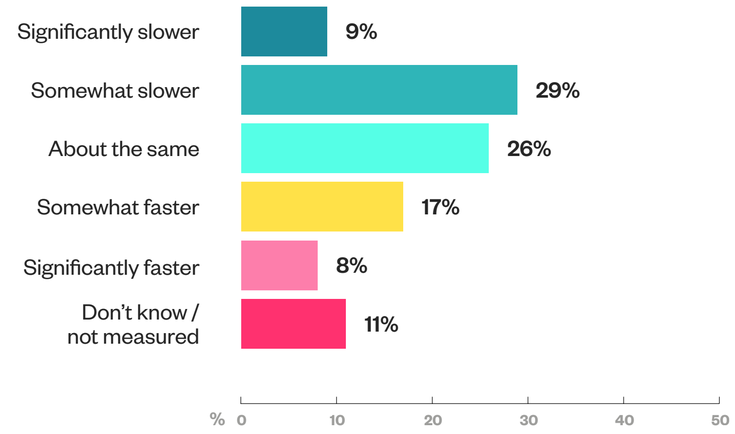
What does this mean?

The extra output from AI coding tools is piling up in review, which means the productivity gain your leadership team expects depends on review capacity you may not have planned for. Start tracking PR volume and review time so you can show them where the work is stuck.

How has the typical size (lines of code) of a pull request changed since AI adoption?



Are code review cycle times faster or slower since AI adoption?



Humans review AI code without enough context

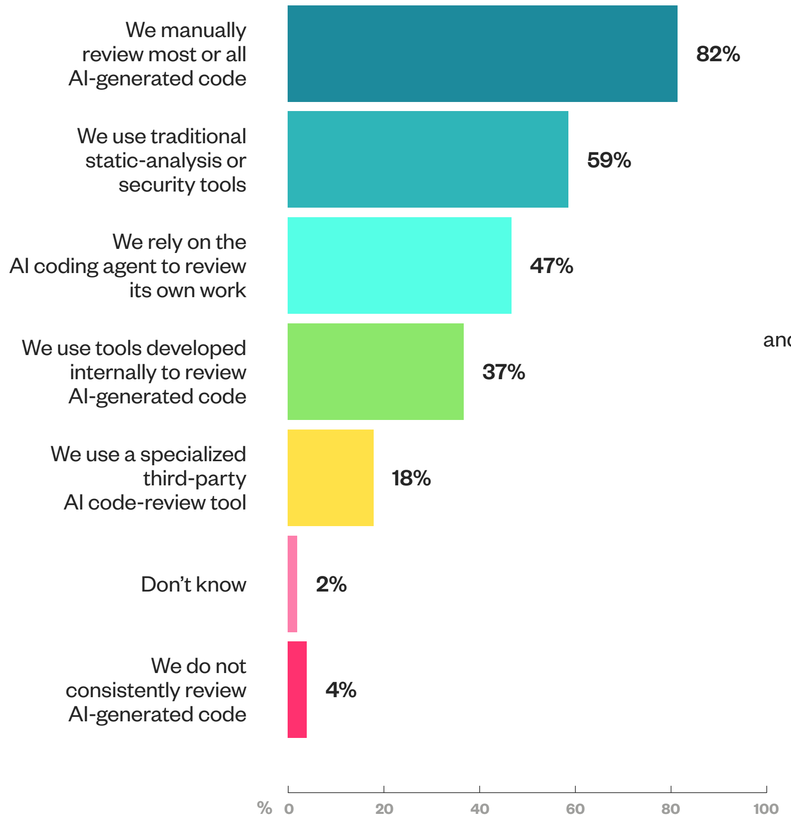


Human review hasn't gone away, with 82% of respondents saying developers manually review most or all AI-generated code before it's merged or deployed. Most teams also use other checks. Asked which ones, and able to pick more than one, 59% name traditional static-analysis or security tools, **47% rely on AI coding agents to review their own work**, 37% use internally developed review tools, and 18% use a specialized third-party AI code-review tool. Only 4% don't consistently review any AI-generated code.

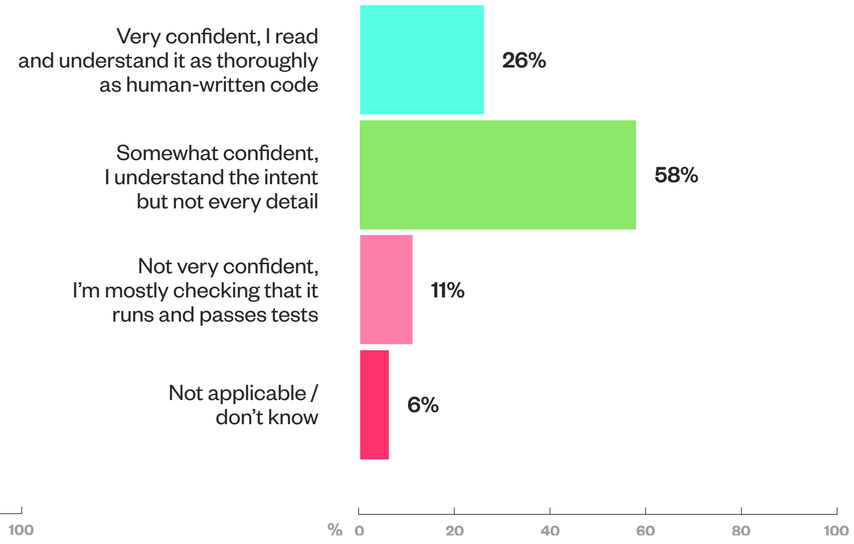
That 47% stat deserves a second look, because nearly half of teams ask an AI agent to check the code it just wrote. Self-review catches obvious slips, but it's a bit like proofreading your own essay straight after writing it. The blind spots that produced the error are the same ones doing the checking.

When respondents personally review a PR they know was AI-generated, **only 26% are very confident they understand it as thoroughly as human-written code**. More than half (58%) understand the intent but not every detail, while 11% admit they're mostly checking that it runs and passes its tests.

Which approaches does your organization use to review AI-generated code before it is merged or deployed?



When you personally review a pull request you know was AI-generated, how confident are you that you fully understand what the code is doing?



Put plainly, roughly seven in ten reviewers check code they don't fully understand.

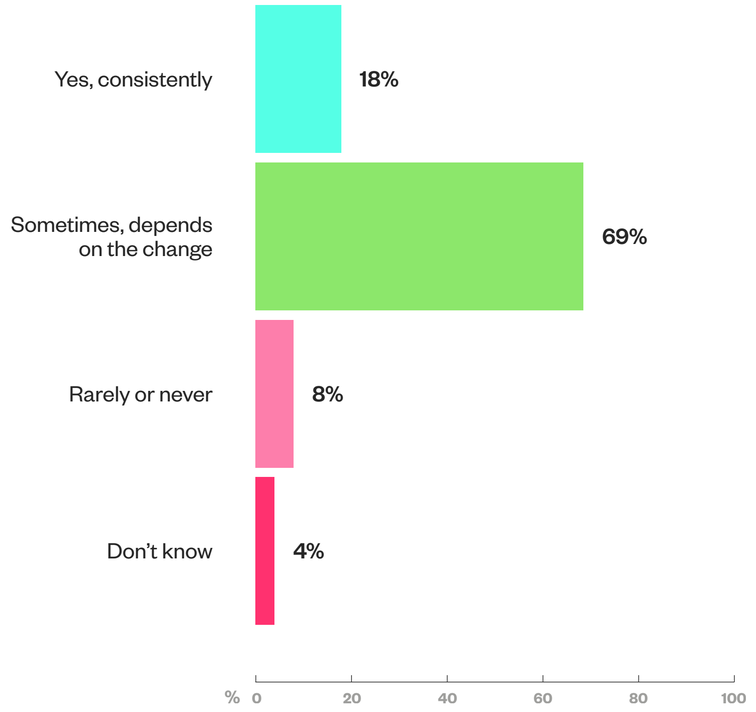
This doesn't reflect their skill level but their workload; the sheer volume of code drives down time available to address each PR.

The healthy counterweight is that, when an AI code review tool flags an issue or approves a change, 74% always or usually have a human independently verify the finding before acting on it (41% always, 33% usually). Only 4% rarely or never check.

The reviewers who understand AI code best are also the most careful: those who are very confident in their understanding are far more likely to work at an organization that always verifies AI review findings (64% vs. 41%). Expertise breeds skepticism, which is exactly what you want in a reviewer. The sharper problem relates to triaging, with 68% of those polled saying reviewers don't consistently know which changes are routine and which need deeper scrutiny before opening a PR. Only 25% flag or triage every change appropriately, while 44% flag only high-risk work such as security-sensitive, customer-facing, or infrastructure-related changes, and 24% flag nothing at all.

Context is just as thin with **only 18% of respondents stating that reviewers regularly have enough information** to understand an AI-generated change's architectural impact, dependencies, and blast radius.

When reviewing AI-generated code, do reviewers typically have enough context to understand its architectural impact, dependencies, and potential blast radius?



This is where the data offers one of its clearest lessons. Organizations whose reviewers consistently have enough context are far more likely to report faster review cycles (40% vs. 25%). Context makes review safer and quicker because reviewers stop burning time reconstructing what a change is meant to do.

The costs of partial understanding show up later: **56% believe AI-generated code is harder to maintain than human-written code,** including 16% who say it is noticeably harder. A quarter (25%) see no difference and only 11% say AI code is easier to maintain. Code that nobody fully understood at review time is code that somebody will struggle with at 2 am during an incident.

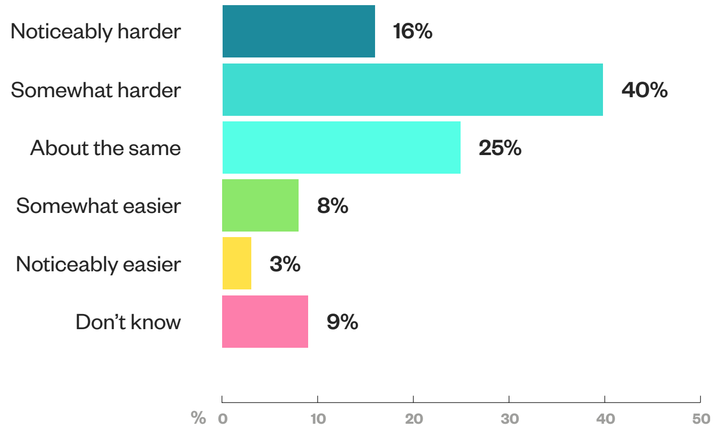
The quieter risk sits with junior engineers. Code review has long been one of the main ways juniors learn from senior colleagues, and AI-assisted review is eating into that. Many respondents (36%) don't know how much impact it has had on the amount of substantive feedback juniors receive. Among respondents with an opinion, 42% say it has decreased and 26% say it has increased.

If senior reviewers are stretched thin and AI handles the first pass, the feedback that used to shape a junior engineer's judgment is the first thing to go. Your senior engineers of 2030 are learning their craft right now, and the data suggests many of them are getting less coaching than their predecessors did.

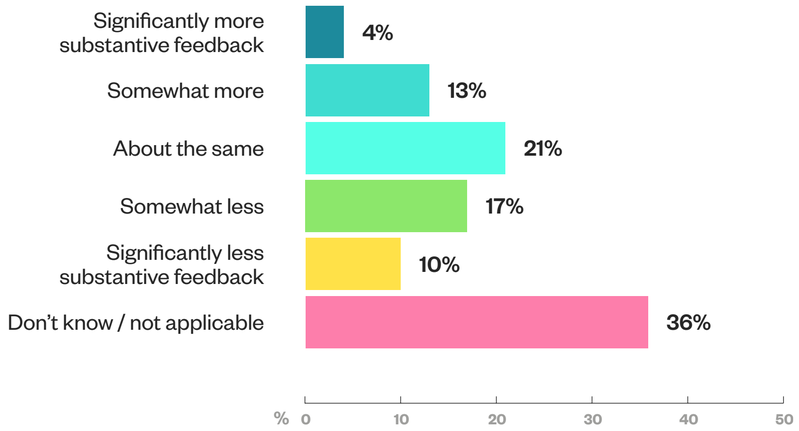
What does this mean?

Most reviewers are approving AI-generated code that they only partly understand, which slows maintenance and reduces the amount of coaching your junior engineers get. Giving reviewers clear triage guidance and context on each change's blast radius makes reviews faster and safer.

How does the maintainability of AI-generated code compare to human-written code, in your experience?



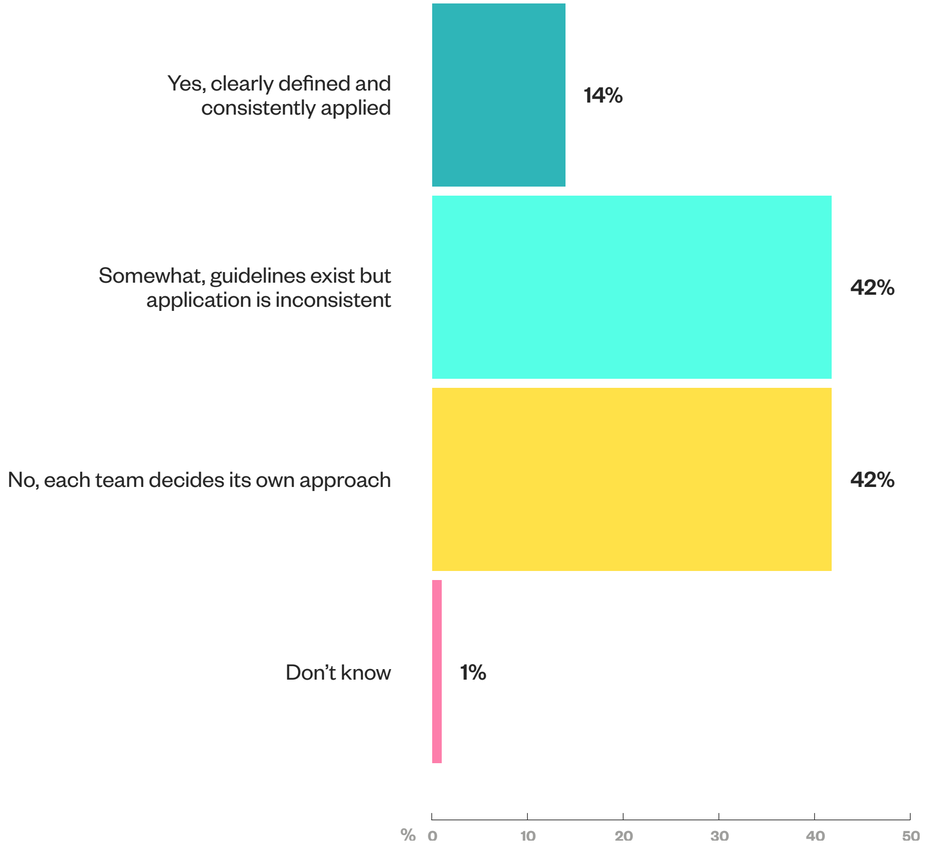
Has AI-assisted code review changed the amount of substantive feedback junior engineers receive from senior colleagues during review?



Governance hasn’t caught up with adoption

AI coding went from experiment to default in a remarkably short time, but the rules around validating its output did not keep pace. Only 14% of organizations have clearly defined, consistently applied standards for reviewing AI-generated code across teams and projects, 42% have guidelines that are applied inconsistently, and another 42% let each team decide its own approach.

Does your organization have consistent standards for reviewing AI-generated code across teams and projects?



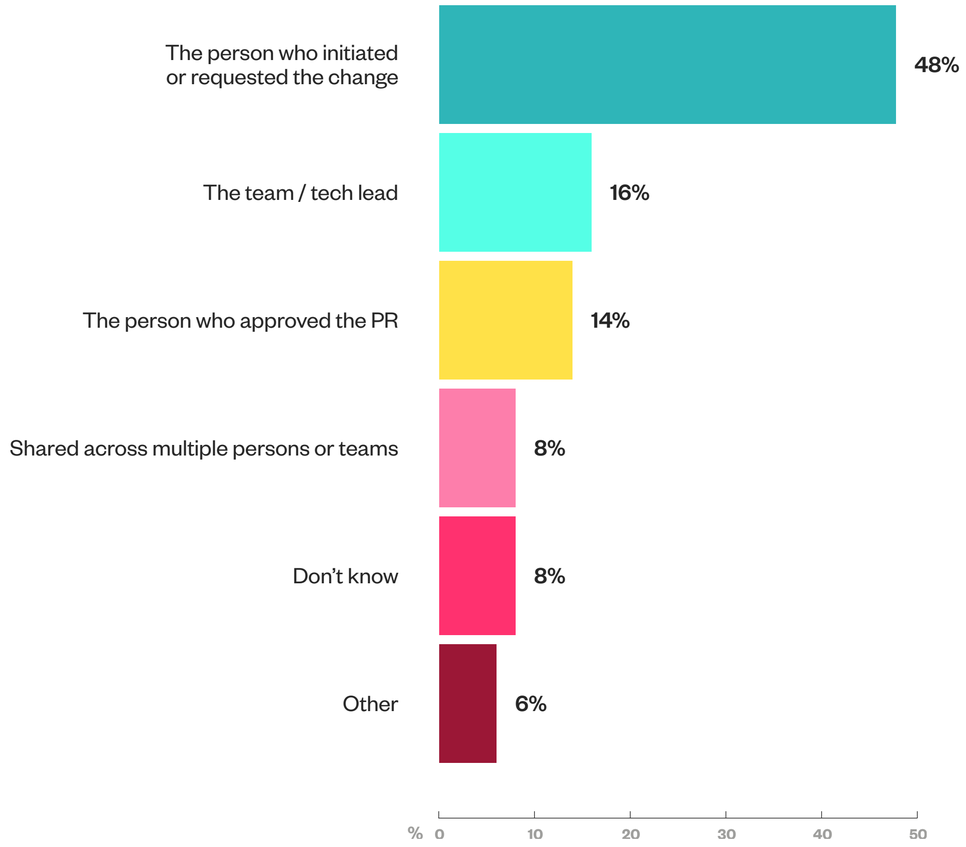
Autonomy helps your team in most areas, except when it lets code reach production that hardly anyone has read. When every team sets its own bar, the weakest bar becomes the organization's real standard, and you only find out where it sits after something breaks.

The survey put a pointed scenario to respondents: an AI agent writes the code, another AI agent reviews it, and an automated system merges it. Who is ultimately accountable for how that change behaves in production?

Almost half (48%) say it's the person who initiated or requested the change, 16% point to the team or tech lead, 14% to the person who approved the PR, and 8% say accountability is shared across people or teams.

Putting responsibility on the requester is a defensible option, and it is also a shift. In a traditional review process, the approver shares ownership of what ships. In this fully automated scenario, only 14% would hold the PR approver accountable, which raises a fair question about what approval will mean as more of your pipeline runs without people. If approval doesn't carry accountability, decide now what it does carry.

If an AI agent writes code, another AI agent reviews it, and an automated system merges it, who is ultimately accountable for that change at your organization?



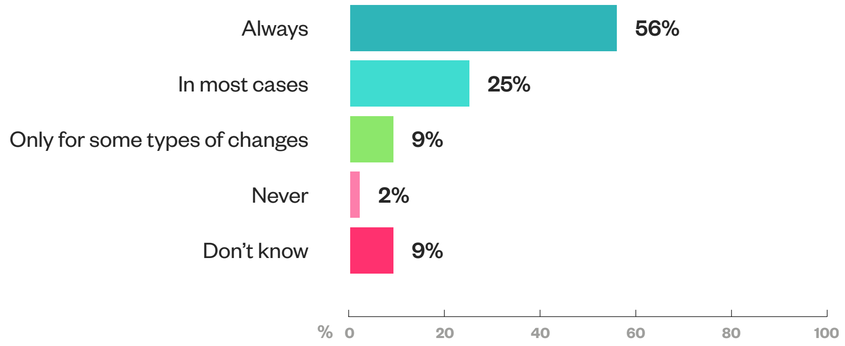
Over half of respondents (56%) say they can always trace every AI-assisted code change back to a specific human owner who approved it. Meanwhile, 25% can do so in most cases, 9% only for some types of change, and 2% never. For regulated industries, and for anyone who has sat through a serious postmortem, “most cases” is not a comfortable answer.

Organizations that flag or triage all changes before review are far more likely to have the ability to always trace changes to an approver (75% vs. 56%). The same habit that tells reviewers where to look also leaves a clean ownership trail behind it. One process change, two problems addressed.

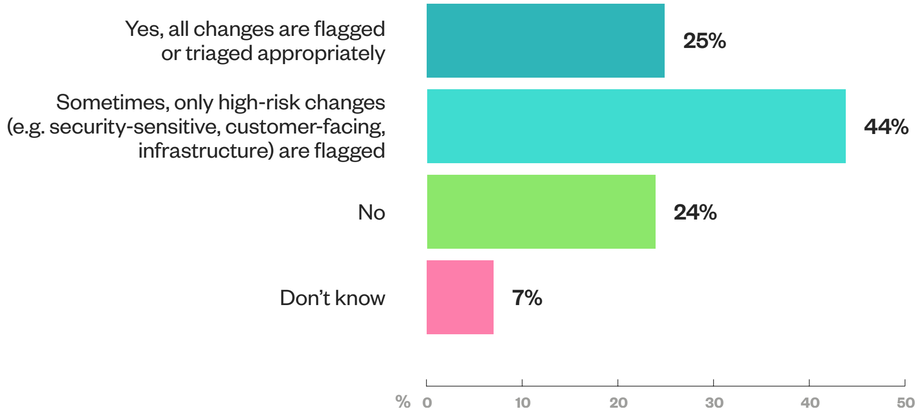
What does this mean?

Most teams set their own review standards for AI-generated code, so your weakest team’s quality bar becomes the company standard, and ownership blurs when things break. **Triaging every change before review shows reviewers where to focus their attention to avoid costly mistakes later on**, and which changes are low-risk enough to be merged without significant oversight.

How often can your organization reliably trace every AI-assisted code change back to a specific human owner who approved it?



Do your reviewers know, before opening a pull request, which changes are routine versus which require deeper scrutiny?



AI review tools help without proving ROI

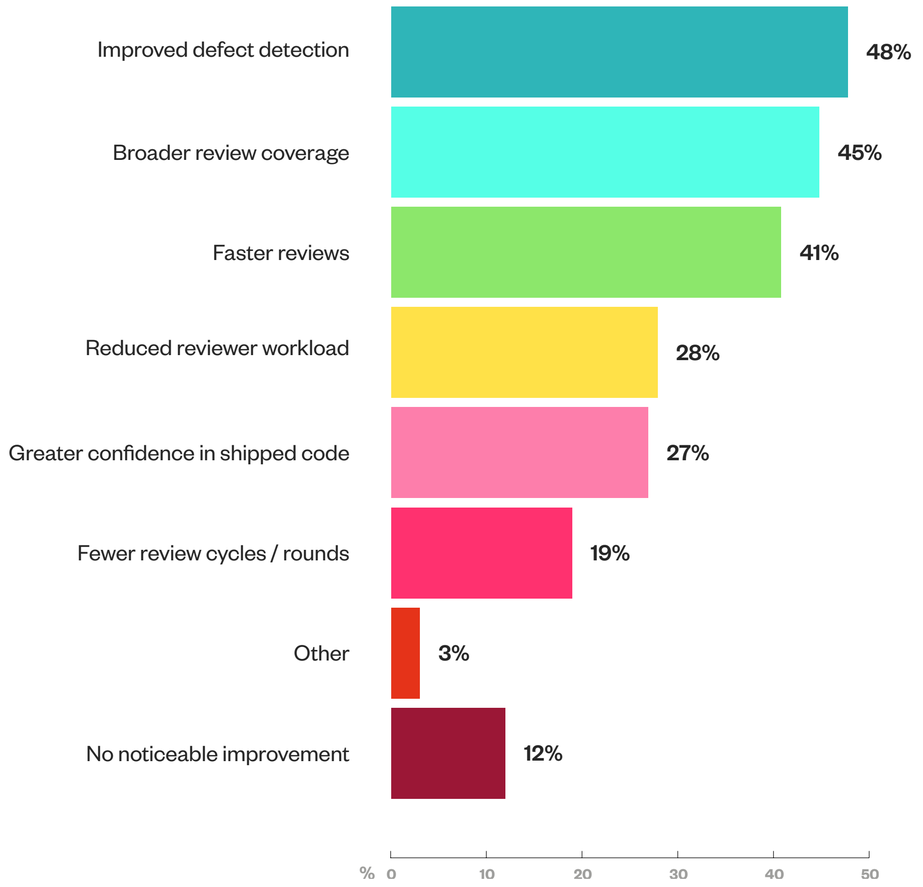
Only 18% of organizations use a specialized third-party AI code review tool, so the market is still young. **CodeRabbit leads with 39%**, almost four times the share of Git AutoReview or Greptile (10% each). Qodo (7%) and Graphite (6%) follow.

Another 36% named a tool not on the list, and GitHub Copilot and Cursor Bugbot made up 43% of those write-ins. CodeRabbit users were also more likely to track review productivity with data (75% vs. 44%), though that comes from just 24 respondents.

The rest of this section covers a wider group: the 88% of organizations that have started to use AI-assisted review, including coding agents that check their own work and internally built tools. Among them, 60% say AI-assisted review produces better outcomes than review without AI, including 47% who say somewhat better and 13% who say significantly better. That's a solid result, as only 9% say outcomes got worse.

The majority (88%) of tool users report at least one improvement. The most common are better defect detection (48%), broader review coverage (45%), and faster reviews (41%). Reduced reviewer workload trails well behind, at 28% of respondents.

Where your organization uses AI code-review tools, what improvements have you observed?



That last number is important, because many teams adopted AI review hoping it would take pressure off overloaded reviewers. For most, it has made review better without making it smaller. The exception is organizations where every team uses AI for all use cases: 45% of them report reduced reviewer workload, compared with 28% overall. Deep, consistent adoption is where AI review starts to save real time.

Respondents were blunt about the downsides, with **70% worrying about false positives and noisy feedback** and 67% finding it hard to trust AI judgment on complex changes. Over half (54%) fear over-reliance on automation, 44% cite cost, and 23% point to a lack of transparency in how issues are flagged. Only 3% have no significant concerns.

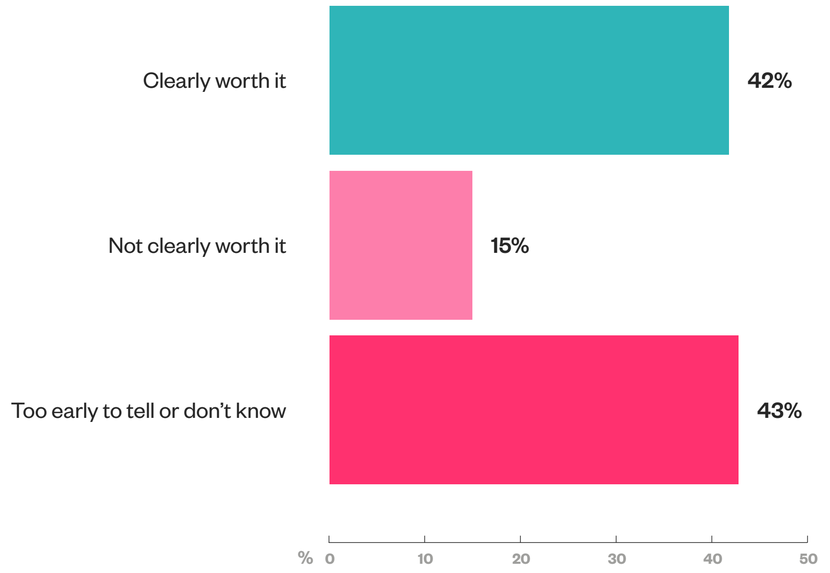
Noise is a real cost. Every false positive is a small tax on reviewer attention, and attention is a resource that is already in short supply.

AI hasn't changed defect rates much in either direction: 19% say escaped defects in AI-assisted code have decreased since AI adoption, while 22% say they've increased. Meanwhile, 42% see no change and 19% don't know.

Excluding those who couldn't answer, 51% say the defect rate is not significantly different. Respondents who did see fewer escaped defects are far more likely to cite greater confidence in shipped code as a benefit (46% vs. 27%).

Weighing time saved against licensing, token usage, and the effort of triaging false positives, 42% say AI code review has clearly been worth it and 15% say it has not. Slightly more (43%) say it's too early to tell or don't know, though that group includes the 12% of respondents who don't use AI review tools at all. Those who see clear value are more likely to report fewer escaped defects (30% vs. 19%), which suggests quality, rather than speed, is what convinces people.

Weighing the time saved against the cost of AI code-review tools (e.g. licensing, token usage, false-positive triage), how would you characterize the return so far?



The ROI split makes more sense once you see how little most organizations measure: 49% track fewer than three code review metrics and 24% track none at all.

Throughput gets the most attention with 48% tracking review or cycle time, 47% tracking PR volume, 18% tracking review rounds per PR, and 62% tracking at least one of these. Quality and people metrics lag far behind: 25% track change-failure and PR reversion rates, 24% security findings, 22% escaped defects, 14% rework rate, and 12% defects found before merge.

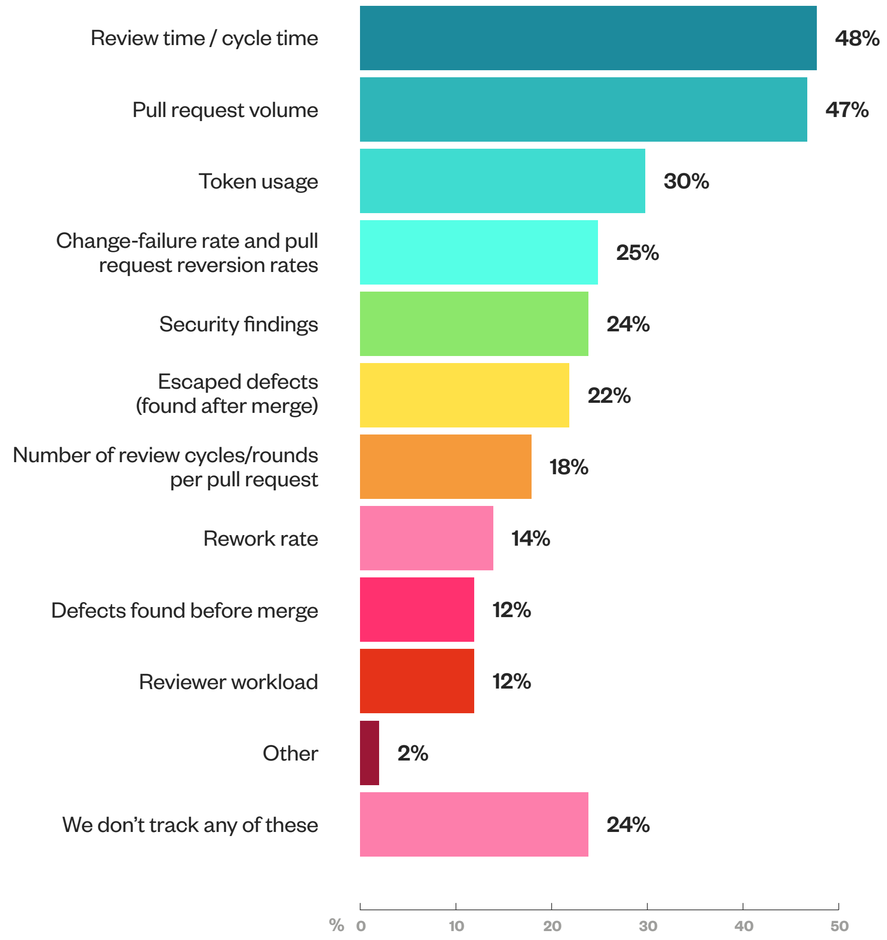
More organizations track token usage (30%) than escaped defects, and just 12% track reviewer workload, even though 45% say review capacity is their biggest bottleneck. Most teams are watching the cost meter and the speedometer while the engine overheats.

When it comes to proving productivity gains from AI-assisted review, 12% of respondents rely only on actual measured data, 32% use a mix of data and perception, 26% rely mostly on perception or anecdote, and 18% haven't made any claims either way. If asked to justify AI review spend next budget cycle, most teams would be doing it on gut feel.

What does this mean?

Close to half of teams can't say yet whether AI review pays off. That gap won't close by itself. Teams that decide now what "worth it" means will have an answer when the budget question comes.

Which of the following does your organization currently track related to code review?



Conclusion

AI has changed where the hard work in software delivery sits. Writing code is cheap and fast. Understanding it, validating it, and maintaining it are not.

The organizations in this survey that are coping best haven't found a magic tool. Instead, they've just changed how review works around the tools they already have.

Here are six ways to put these findings into practice:

- **Plan for review capacity**.
  With 45% naming review as their biggest bottleneck, review time belongs in sprint planning and headcount conversations, alongside build time and on-call load.

- **Triage before anyone looks at a PR**.
  Only 25% appropriately flag or triage every change. Those that do are far more likely to be able to always trace a change to its approver (75% vs. 56%), allowing reviewers to focus their attention where it counts.

- **Hand reviewers the context.**
  Teams whose reviewers consistently understand impact, dependencies, and blast radius are more likely to see faster review cycles (40% vs. 25%). Link changes to their intent, the tickets behind them and the systems they touch.

Keep PRs small.
When PRs grow significantly larger, 55% see review slow down, against 38% overall. Plan ahead, break down large tasks into smaller chunks, and commit often.

- **Write your team's standards, and update often.**
  Only 14% apply consistent review standards, and accountability defaults to the requester for 48%. Decide what approval means at your organization and write it down before an incident decides it for you.

- **Measure quality and people, too.**
  Throughput is tracked widely, but only 22% track escaped defects and 12% track reviewer workload. Add both, and watch the feedback junior engineers receive, since 42% of those with a view say it has dropped.

These patterns are correlations, so treat them as strong signals rather than proof. Still, they line up with what most experienced engineering leaders already know. Review works when people know what they're looking at, why it changed, and who owns it.

AI will keep producing code faster. Your job is to make sure your organization can still understand, validate, and maintain what ships. Teams that build this capacity now will have the data to show that their AI investment is paying off.

Methodology

The survey was conducted between **29 September and 17 October 2026**. The analysis is based on **377 respondents**, of whom **325 completed the entire survey**. Margin of error: +/- 5% at a 95% confidence level. Due to rounding, some graphs may not add up to 100%.

