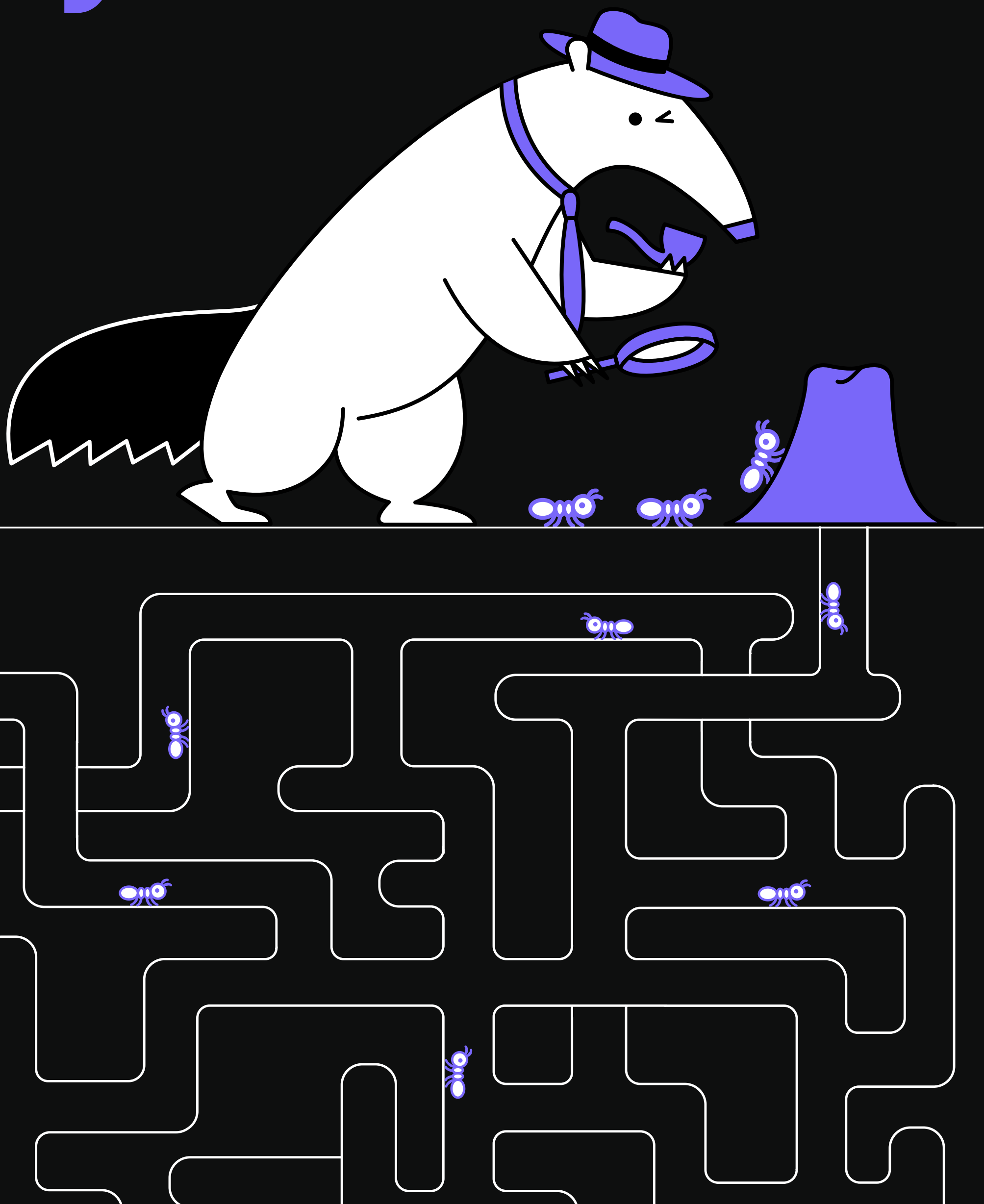
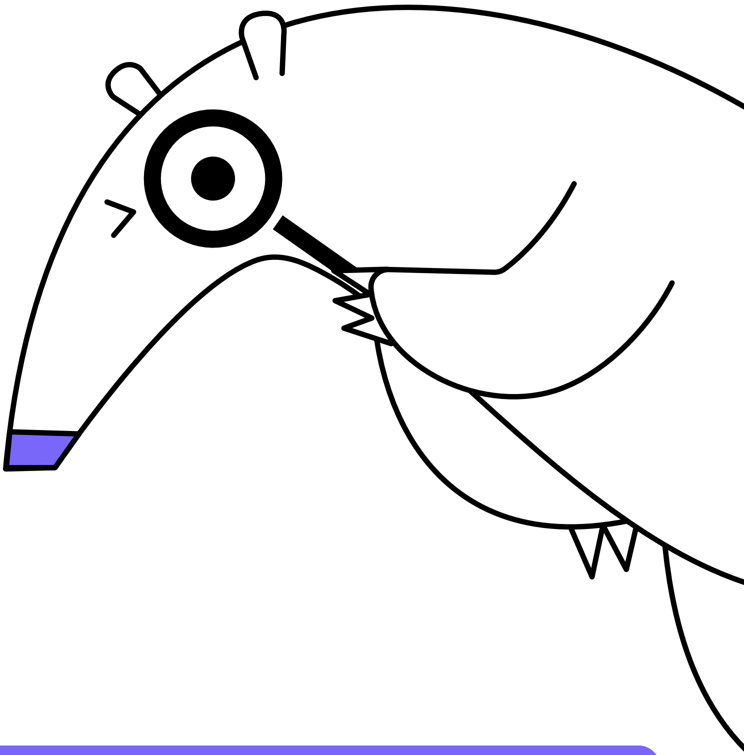
qodo

QODO REPORT

2025 State of AI Code Quality





Introduction

AI coding is no longer judged by how much code it can generate – it is judged by how confident developers are with the code generated.

> AI tools have become a daily fixture in software development, offering clear productivity boosts and even enhancing the enjoyment of the work. Yet, despite their widespread use, a deep trust in their output remains elusive. Developers aren't satisfied with code that merely compiles – they want intelligent suggestions that are context-aware, adhere to conventions, and are backed by tests. Without this, AI-generated code still demands heavy human oversight, and that undermines its promise of efficiency.
>
> To close this trust gap, AI must become more than just an autocomplete engine. It needs to be embedded deeply within the development lifecycle – as an always-on, context-aware reviewer that proactively supports code quality and consistency. This is a call for both toolmakers and engineering teams to prioritize trust alongside productivity by investing in continuous review, automated testing, and integrated context awareness. Only with that foundation can AI truly transform software development.

Itamar Friedman

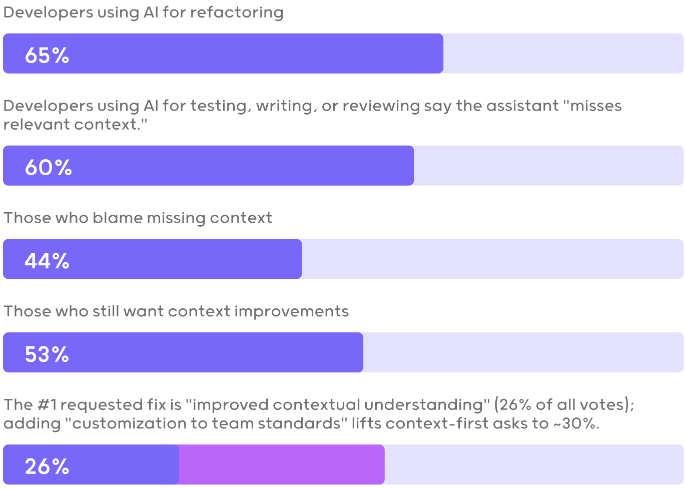
Co-founder & CEO, Qodo

Executive Summary: In a 2025 survey of 609 developers across company, sizes and regions, three themes emerged.

1 Context is the foundation of trust

- 65% of developers using AI for refactoring and ~60% for testing, writing, or reviewing say the assistant "misses relevant context."
- Among those who feel AI degrades quality, 44% blame missing context; yet even among quality champions, 53% still want context improvements.
- The #1 requested fix is "improved contextual understanding" (26% of all votes); adding "customization to team standards" lifts context-first asks to ~30%.

Implication: A learned, repo-wide context engine is not a nice-to-have–it is the foundation for accuracy, quality, and trust across the SDLC.


Developers using AI for refactoring

Developers using AI for testing, writing, or reviewing say the assistant "misses relevant context."

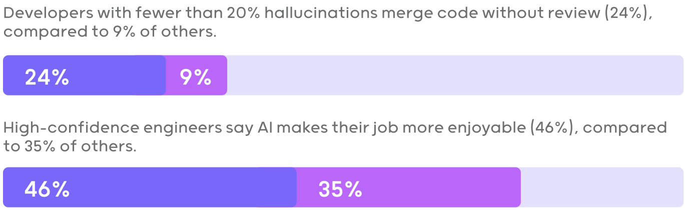
Those who blame missing context

Those who still want context improvements

The #1 requested fix is "improved contextual understanding" (26% of all votes); adding "customization to team standards" lifts context-first asks to ~30%.

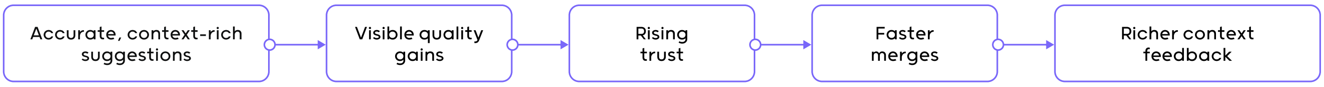
2 Confidence is the key to adoption

- Developers who experience fewer than 20% hallucinations are 2.5x more likely to merge code without reviewing it (24% vs. 9%).
- High-confidence engineers are 1.3x more likely to say AI makes their job more enjoyable (46% vs. 35%).


Developers with fewer than 20% hallucinations merge code without review (24%), compared to 9% of others.

High-confidence engineers say AI makes their job more enjoyable (46%), compared to 35% of others.

- The "Confidence Flywheel":

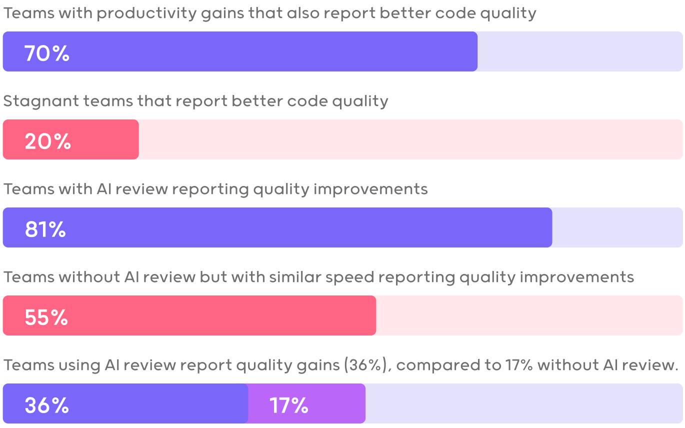


Implication: Any AI workflow that cannot prove its accuracy will stall adoption long before reaching scale.

3 Productivity and quality rise together – when review is automated

- When teams report "considerable" productivity gains, 70% also report better code quality – a 3.5x jump over stagnant teams.
- With AI review in the loop, quality improvements soar to 81% (vs. 55% for equally fast teams without review).
- Even without a boost in delivery speed, teams using AI review see double the quality gains (36% vs. 17%).

Implication: Continuous, opinionated review is the force-multiplier that converts raw speed into durable quality.


Teams with productivity gains that also report better code quality

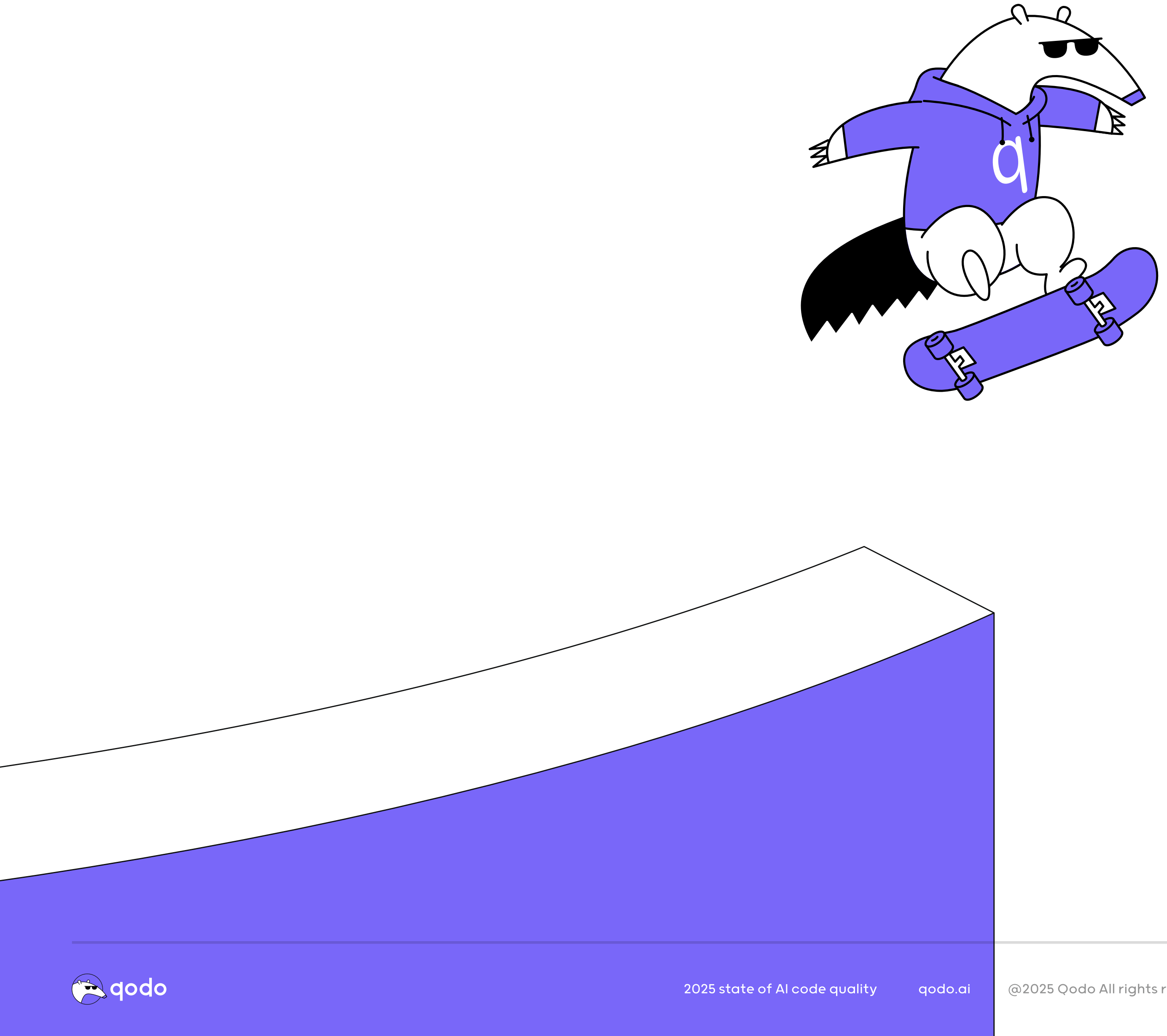
Stagnant teams that report better code quality

Teams with AI review reporting quality improvements

Teams without AI review but with similar speed reporting quality improvements

Teams using AI review report quality gains (36%), compared to 17% without AI review.

In this report:



Part 1: State of AI coding adoption

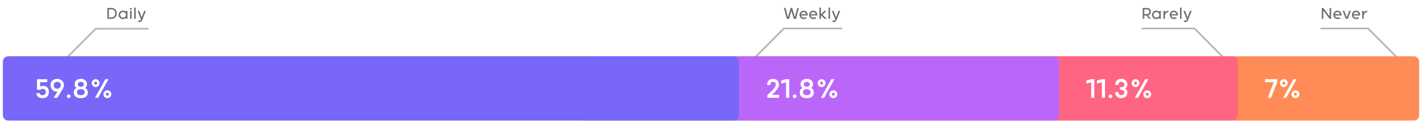
AI assistance is no longer a curiosity on the developer desk; it is a mainstream, multi-tool layer woven through the entire software lifecycle. What was once experimental is now routine, with developers across team sizes and industries integrating AI into core workflows.

This section highlights how deeply embedded these tools have become, how developers perceive their impact, and where the current friction points remain. The data reveals not just adoption, but reliance-and hints at how AI is reshaping the way code gets written, reviewed, and shipped.

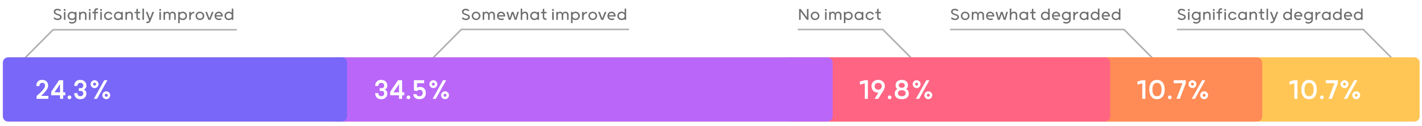
Here are some key points we learned:

- 82% of developers use AI coding tools daily or weekly
- 59% run three or more tools in parallel
- 65% say AI touches at least a quarter of their codebase
- 78% report productivity gains
- 57% say AI makes their job more enjoyable

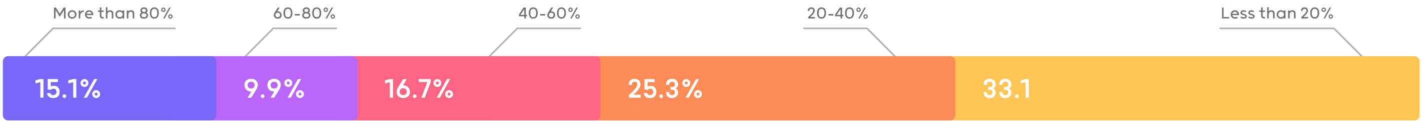
How frequently do you use AI coding tools in your workflow?



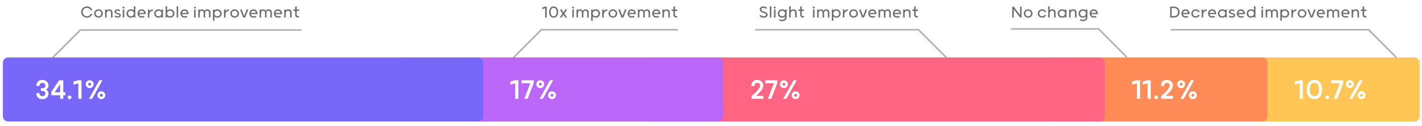
Has AI improved or degraded your overall code quality?



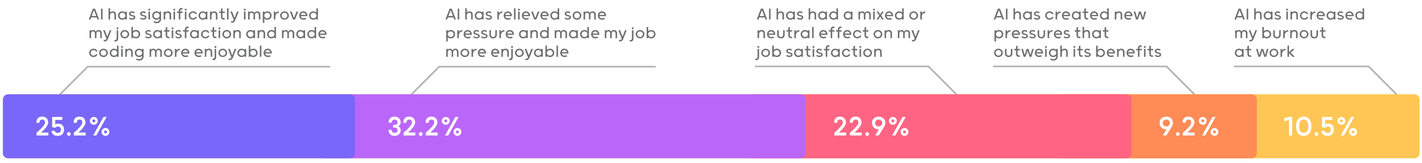
What % of your code is currently generated or significantly influenced by AI tools?



Has AI made you more or less productive?



Which statement best describes how AI has impacted your job satisfaction?

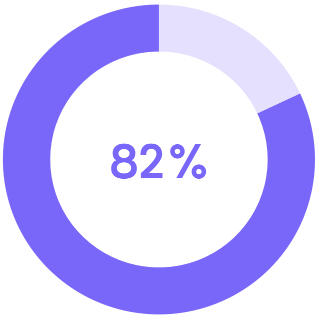


Adoption is broad, frequent, and multi-tool

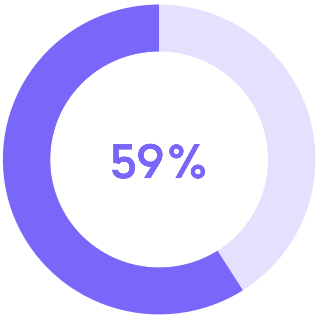
Most developers aren't just trying AI coding tools–they're relying on them:

- 82% say they use an AI coding assistant daily or weekly, a clear sign that these tools have moved from experimentation to core workflow.
- 59% use three or more AI tools regularly, and 20% manage five or more – a signal of both enthusiasm and fragmentation in the tool landscape.
- Dev teams juggling 6+ tools lack shipping confidence – just 28% are sure of their AI code.

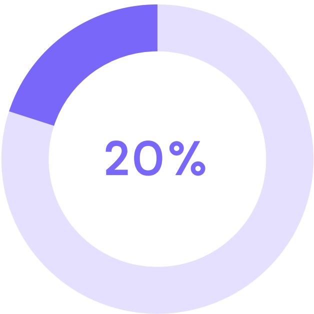
This multi-tool reality reflects how AI is being woven into many parts of the SDLC, from code generation to explanation, refactoring, and testing.



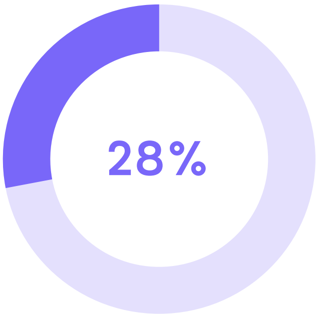
82% of developers use AI coding tools daily or weekly



59% use three or more AI tools regularly



20% manage five or more AI tools regularly



28% of developers are sure of their AI code

Adoption is especially strong in smaller teams:

- 51% of active AI users are in companies with 10 or fewer developers.
- Having said that, 25% of enterprises with 100+ engineers are also past the experimentation stage–showing that scale isn't a blocker to meaningful use.

Industry snapshot

Adoption clusters in Tech & Software (56% of respondents) but spreads across Finance (9%) and beyond, reflecting sector-agnostic demand for faster, higher-quality development cycles. The smallest startups move quickest, but larger organisations are closing the gap as governance patterns mature.

AI Is influencing real code, not just prototypes

AI-generated code is making its way into production–not just pull requests:

65% of developers say at least **a quarter** of each commit is generated or shaped by AI.

15% say that **more than 80%** of their code is now AI-influenced.

That means "AI-touched" code is fast approaching parity with traditional development in many teams - and suggests that GenAI is becoming a silent contributor to production systems.

Developers see gains in both speed and quality

Although there are concerns about accuracy, developers generally report positive outcomes from using AI tools:

- 59% say AI has improved code quality, while 21% report degradation
- 78% report productivity gains, with 17% claiming a "10x" increase in output
- 57% say AI makes the job more enjoyable and helps reduce pressure

These responses suggest that AI isn't just changing how fast developers work–but how they feel about their work, too.

Hallucinations are still holding teams back

25% of developers estimate that 1 in 5 AI-generated suggestions contain factual errors or misleading code.

Widespread adoption hasn't eliminated one of the biggest blockers to AI reliability: hallucinations.

For some, these are acceptable nuisances. For others, they're a hard-stop–especially when working in a production environment or secure code bases.

We'll uncover in this report, how trust in AI output is directly tied to how accurate, contextual, and reviewable AI generated code is.

Part 2: AI Code Quality across the SDLC

As GenAI becomes a regular part of software development, the question is no longer "Can it generate code?"– but "Is the code good, and do developers trust it enough to use it?" Speed and volume are easy to track, but they don't reflect whether AI is actually improving software or helping teams move faster with less friction.

That's why we focused this part of the report on two essential metrics: code quality and developer confidence. Quality reflects the outcome – how clean, accurate, and production-ready the code is. Confidence reflects the experience – whether developers trust the AI enough to rely on it during real work: merging code, writing tests, or reviewing changes.

Together, these signals tell us whether GenAI is just generating output–or actually driving value.

Velocity ≠ Corner-Cutting: higher productivity goes hand-in-hand with higher code quality

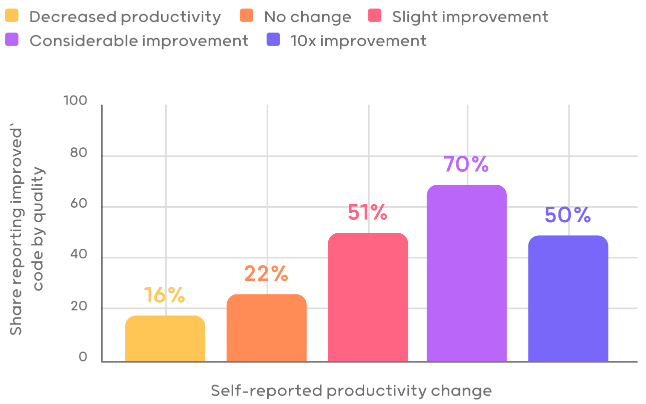
A common fear with AI tooling is that faster delivery comes at the cost of quality. Our data shows the opposite. When AI meaningfully improves developer productivity, code quality improves right alongside it.

Speed and quality rise together

According to our survey:

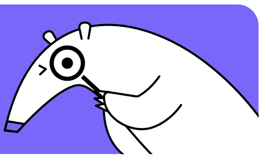
- 70% of developers who saw considerable productivity gains also reported improved code quality
- That's a 3.5x increase over those whose productivity stayed the same or declined (16-22%)
- Even a slight productivity bump led to 51% reporting better quality

Productivity vs Perceived code-quality impact of AI



This shows that AI impact isn't confined to low-stakes boilerplate. When speed rises, so does trust in the code. That's because quality doesn't come from speed alone – it comes from how speed is achieved.

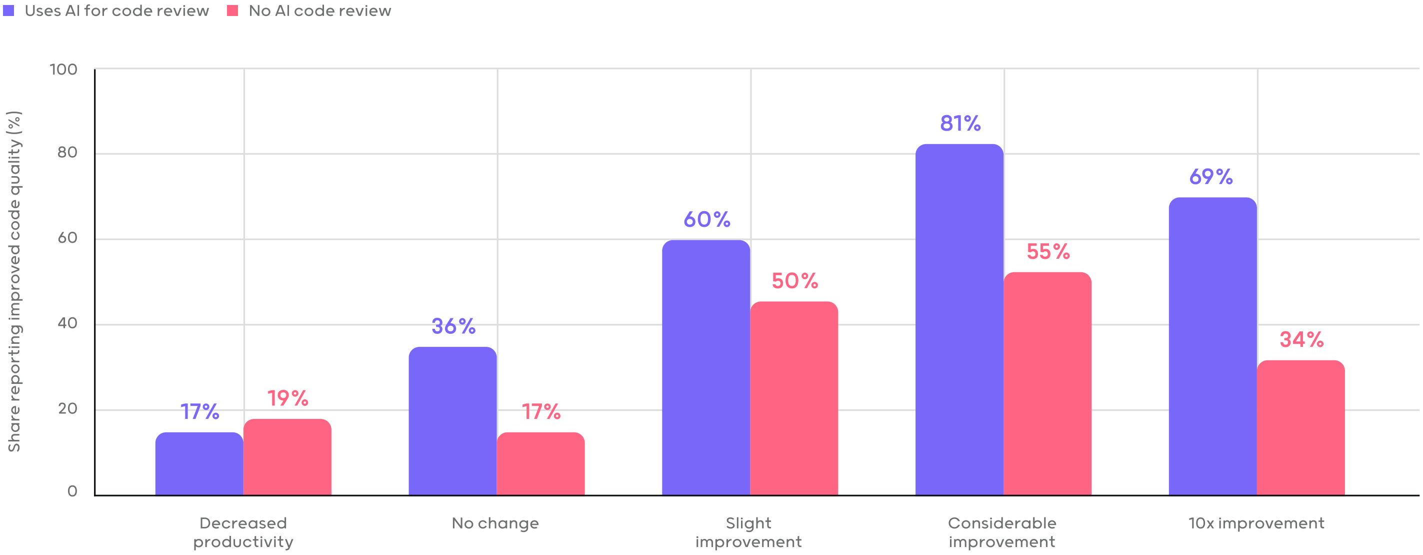
Qodo product insight: Issues of high severity (score 9-10) were identified in 17% PRs.



Continuous review: the catalyst that turns raw speed into higher quality

The data from the survey shows a clear pattern: AI-powered code review is where productivity gains are converted into real, measurable quality improvements.

Quality gains vs Productivity (review adopters vs non-adopters)



Among developers reporting considerable productivity gains:

- 81% of those who use AI for code review saw quality improvements
- Just 55% of fast-moving teams without AI review saw the same

Even when productivity didn't change:

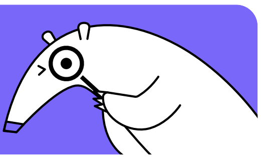
- 36% of AI-review adopters reported quality gains
- That's double the 17% seen among non-reviewers in the same productivity band

And for teams claiming a "10x improvement" in speed:

- 69% with AI review said quality improved
- Compared to just 34% without review

These findings reinforce that continuous review is a missing link between speed and quality. It delivers an early win – lifting code quality even before throughput rises – and scales to guardrail the teams that push velocity to the limits.

Qodo product insight: When an AI-review tool is enabled, 80% of PRs don't have any human comment or review.



State of developer confidence in AI coding

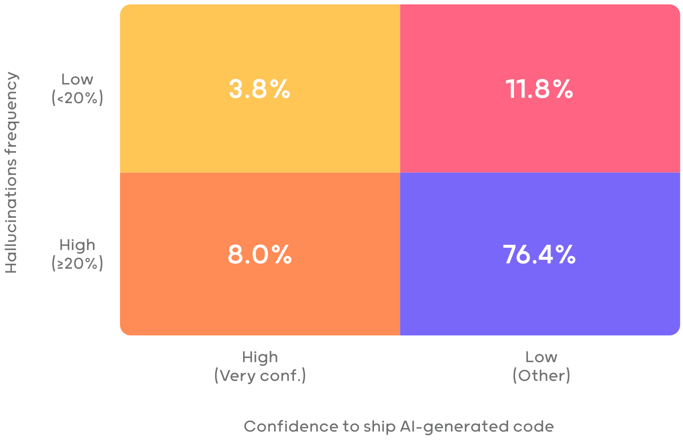
The confidence - hallucination divide

To understand how developers perceive the reliability of AI-generated code, we mapped their responses across two axes:

Hallucination frequency: how often the AI produces incorrect or misleading code.

Confidence: how comfortable developers are shipping AI code without human review.

Hallucinations vs Shipping-Confidence



Each quadrant represents a different behavioral profile:

Top Left (3.8%)
Low hallucinations, high confidence

- The ideal scenario. These developers trust AI and see accurate output. But they're a small minority - just 3.8% fall into this quadrant.

Top Right (11.8%)
Low hallucinations, low confidence

- Even when output is accurate, most developers are still cautious. Three out of four low-hallucination users don't fully trust what AI generates.

Bottom Left (8.0%)
High hallucinations, high confidence

- These developers use AI despite frequent mistakes–possibly because they review everything manually, or apply it only in low-risk areas.

Bottom Right (76.4%)
High hallucinations, low confidence

- This is the dominant group. More than three-quarters of developers encounter frequent hallucinations and avoid shipping AI-generated code without human checks. This quadrant represents a significant drag on AI adoption and ROI.

Confidence in tests – do AI-testing adopters trust their tests more?

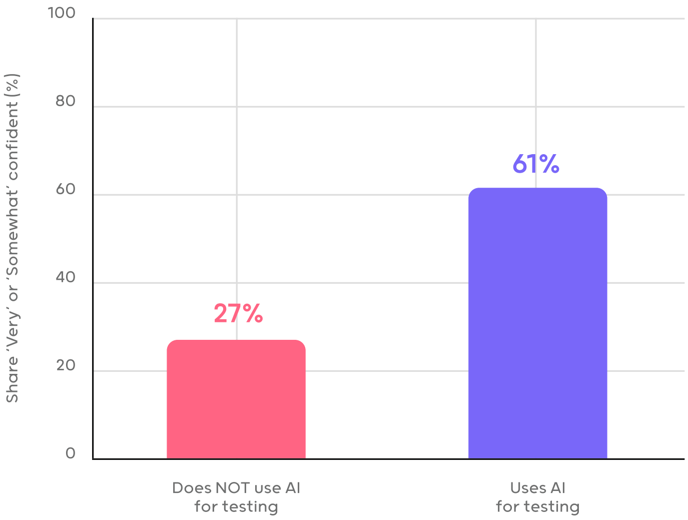
We asked developers how confident they feel in the tests protecting their code – and whether they use AI to generate those tests. The survey uncovered that teams that actually let AI write their tests don't just save time – they end up more than twice as confident in those tests.

Our survey showed that out of all the developers that don't use AI for testing, only 27% of them are "very" or "somewhat" confident in the safety net their tests provide.

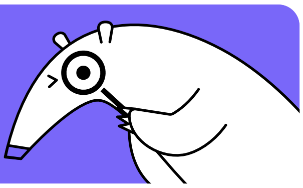
However, adoption is still the exception, not the rule. This confidence gap – a 34-point spread between adopters and non-adopters – is a missed opportunity. Many teams are held back by early experiences with generic, disconnected test generation that lacked awareness of project context or team standards.

But as AI testing becomes more integrated – leveraging real code, referencing existing test structures, and aligning with how teams already work – developers are more likely to trust and adopt it. The shift from isolated test scripts to context-rich, in-flow test generation is what transforms AI from a shortcut into a meaningful quality layer.

Confidence in AI generated tests: Users who do vs. don't use AI for testing



Qodo product insight: When user provides test example file when generating tests with Qodo Gen, tests adoption jumps by 15% in comparison to the cases with no examples.



The confidence gap is huge

76% of developers fall into the "red zone" – frequent hallucinations and low confidence – highlighting a major barrier to AI's value.

These developers are using AI tools, but they don't trust the results. As a consequence, they:

- Manually review or rewrite most suggestions
- Delay merges even when AI-generated code looks correct
- Avoid deeper integration of AI into their workflows

Lack of trust in AI tools undermines their promised productivity gains – teams recheck, discard, or rewrite code and often see limited ROI. This isn't just a technical issue; it's an adoption risk. Low confidence reduces usage and feedback, creating a vacuum that hinders improvement. Adoption tends to be stronger in smaller teams, where trust and iteration move faster.

Why accuracy isn't enough

We found that developers who rarely encounter hallucinations are 2.5x more likely to be very confident in shipping AI-generated code (24% vs. 9%). But even among the low-hallucination group, most developers (75%) will hesitate to merge without manual checks. Why?

Because hallucination-free output isn't the full story - developers want code that:

- Matches project context and architecture
- Aligns with style and conventions
- Is paired with automated checks and coverage

Without that, even "correct" suggestions feel risky.

Part 3: Context - the foundation of trust in AI coding

Developers are starting to use AI across the SDLC, but the most common complaint isn't hallucination-it's relevance. Despite progress in reducing hallucinations, our data shows that AI code suggestions-rooted in context are the top barrier to AI trust levels and code quality.

- 65% say AI misses context during refactoring
- 60% report similar issues during test generation and review
- 44% of those who say AI degrades quality blame context gaps
- Even among AI "champions," 53% want better contextual understanding

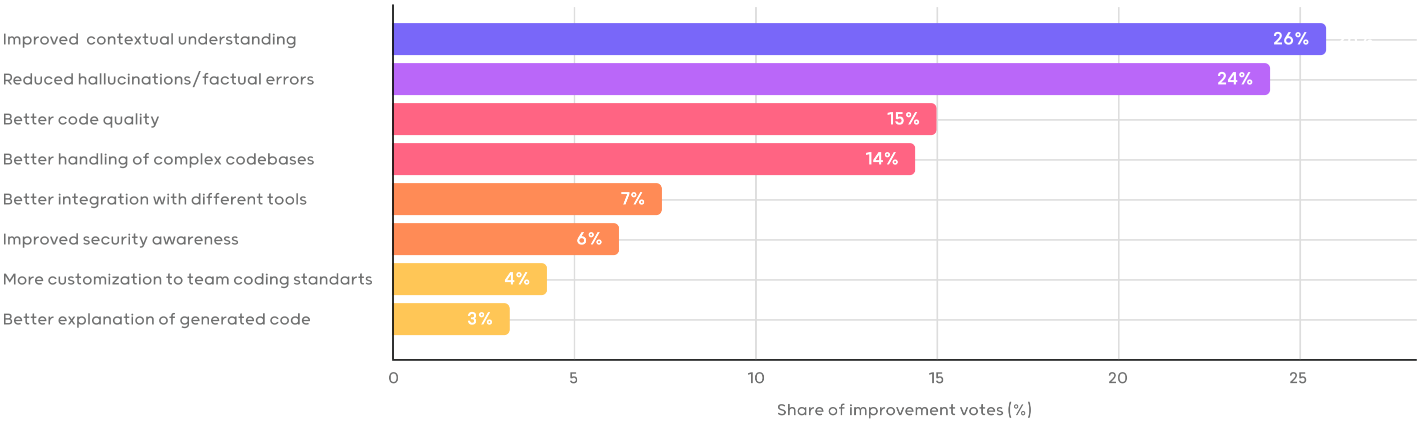
These numbers suggest that contextual intelligence – not just code generation – is now the primary driver of perceived quality.

How does AI tooling sprawl factor into context struggles? Dev teams juggling six or more AI tools still feel context-blind 38% of the time.

Where developers want context to work harder

When we asked developers experiencing "context pain" what they most want from their AI tools, one answer stood out: richer contextual understanding.

Requested Improvements among Context-Struggling Developers



- 26% of all top-three improvement votes focused on "improved contextual understanding"
- This narrowly edged out the next most common request: "reduced hallucinations" (24%)
- Another 4% asked for "customization to team coding standards," which is often rooted in contextual mismatches-not just style enforcement

Taken together, nearly one-third of all improvement requests were about making AI tools more aware of the codebase, team norms, and project structure – well ahead of requests for better quality output (15%) or improved handling of complex systems (14%).

These responses point to a deeper insight: hallucinations and quality issues often stem from the same issue–poor contextual awareness.

When AI suggestions ignore team patterns, architecture, or naming conventions, developers end up rewriting or rejecting the code – even if it's technically "correct." Fixing this doesn't just reduce errors; it unlocks faster code merges, higher trust in AI, and smoother developer collaboration.

Who feels context pain the most?

By company size:

- 50% of developers who say AI misses relevant context work at startups with 10 or fewer employees
- While large orgs face complexity at scale, smaller teams feel context gaps more acutely–likely because every mistake costs more time when resources are limited

By developer seniority:

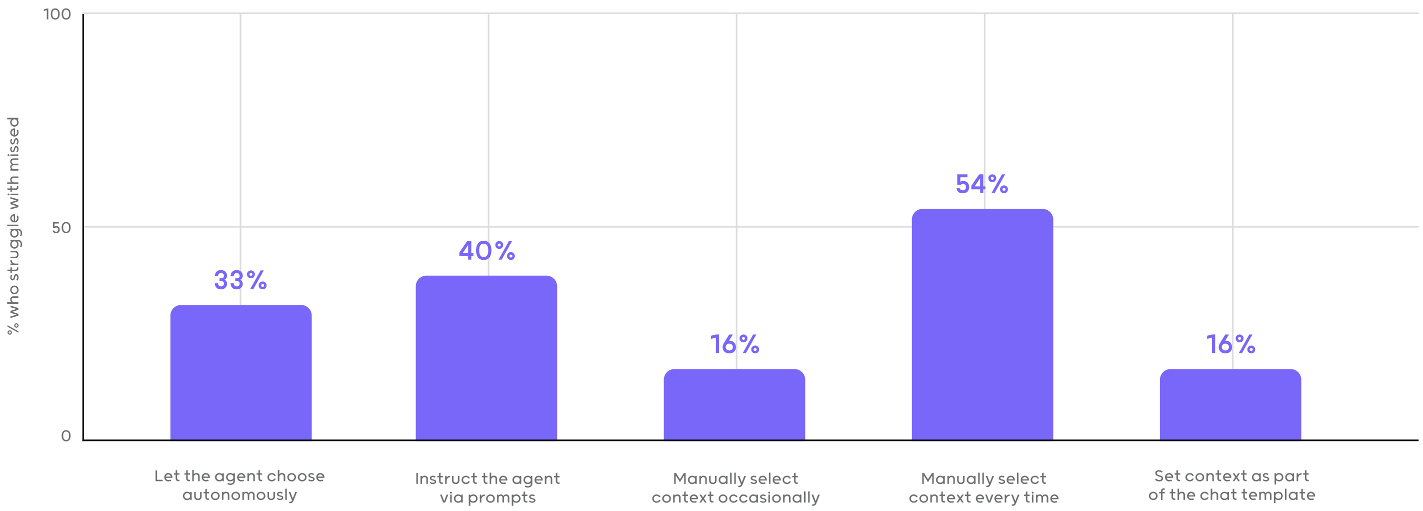
- Context pain increases with experience: **from 41% among junior developers to 52% among seniors.** Senior engineers have deeper mental models of their codebase, and they expect AI to reflect that nuance
- While seniors see the largest quality gains from AI (60%), they also report the lowest confidence in shipping AI-generated code (22%)

Manual context is broken, persistent learning is the fix

Manually selecting context for every prompt – files, functions, folders – might have worked in early tools, but it doesn't scale. It's tedious, error-prone, and leads to frustration when results still miss the mark.

- 54% of developers who manually select context say the AI still misses relevance
- That frustration drops to 33% when tools choose context autonomously
- And falls even further – to 16% – when context is persistently stored and reused across sessions

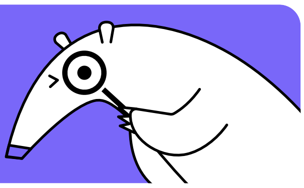
Context-Miss strugle vs Context-Selection method



In short: the more context is automated and retained, the more useful and trustworthy AI becomes.

The shift isn't just about convenience – it's about enabling AI to behave like a teammate who knows your codebase, not a temporary helper who needs constant reminders. Tools that index the full repo, learn team conventions, and remember architectural patterns can deliver suggestions that feel native to the project – and immediately relevant.

Qodo product insight: Agentic chat utilizes an average of 2.5 MCP tools per user message. The majority of tool usage (60%) is dedicated to context identification and content retrieval operations (RAG).



LLM context gaps lurk in every task and they shape how developers judge quality

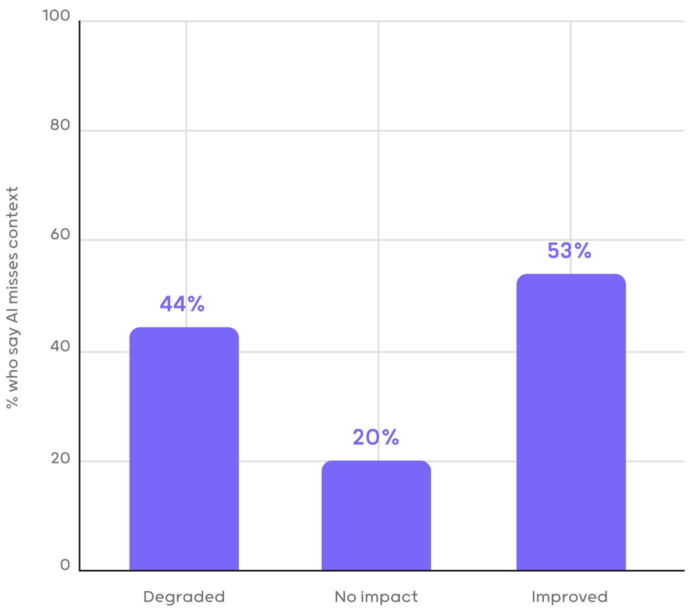
Context gaps aren't limited to edge cases-they emerge in the most common and critical development tasks. Developers are consistently reporting that GenAI tools fail to grasp enough of the codebase to be reliable:

- 65% say AI misses context during refactoring
- 64% during boilerplate generation
- 60% during core tasks like writing or testing
- ~58% during code reviews and explanations

Structural tasks suffer most: Refactoring tops the chart for AI missing context.

In other words: the more a task depends on understanding the broader codebase, the more likely AI is to miss the mark. These aren't fringe issues–they strike where developers expect the most value.

Context pain vs Perceived code quality

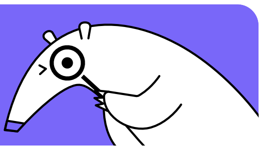


Missing context doesn't just lead to bad suggestions–it distorts how developers assess AI performance overall:

- Among those who say AI degraded quality, 44% blame missing context
- Even among those who say AI improved quality, 53% still want better context

This shows that context is the floor and the ceiling: without it, quality drops; with it, expectations (and scrutiny) rise. A persistent, code-indexing engine is the only path to eliminating those blind spots and unlocking quality gains across every task.

Qodo product insight: 8% of code review suggestions focus on aligning pull requests with established company best practices.



Style mismatches are more than an annoyance–they break trust

When AI-generated code doesn't match a team's coding conventions, trust and productivity suffer. To understand how this affects developer satisfaction, we examined responses across consistency tiers–how often AI suggestions align with team coding style and standards.

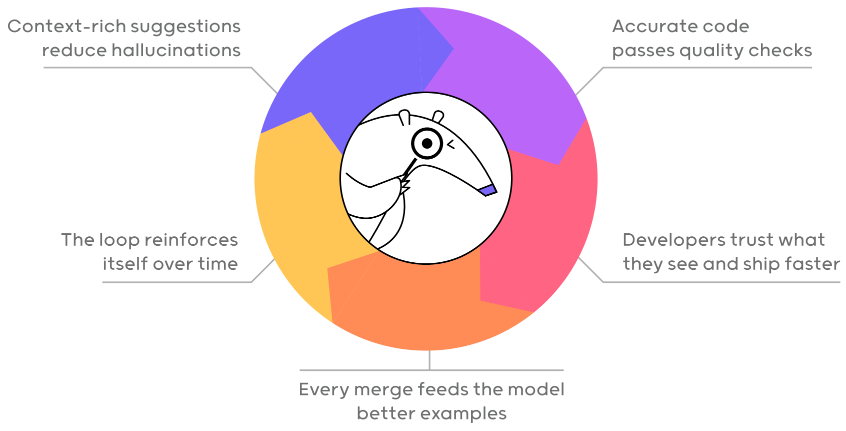
The real insight emerges when we flip the lens:

- Developers who receive inconsistent output are **1.5x more** likely to flag "code not in line with team standards" as a top frustration
- Nearly **two out of five developers** who rarely or only occasionally see style-aligned suggestions cite this as a major blocker

For these developers, every AI assist becomes a fix-it task–slowing momentum, breaking flow, and eroding confidence.

Part 4: The Confidence Flywheel

Our survey reveals a self-reinforcing cycle we call the "Confidence Flywheel"



Context-rich suggestions reduce hallucinations
Accurate code passes quality checks
Developers trust what they see and ship faster
Every merge feeds the model better examples
The loop reinforces itself over time

It starts with grounded suggestions: when an AI assistant pulls precisely the right code context – local files, architectural patterns, team conventions – the rate of hallucinations plummets.

Fewer errors translate into visible quality gains as developers see cleaner diffs, higher test coverage, and far fewer nit-pick reviews. Those tangible wins build personal trust, so teams begin shipping AI-generated code faster and with less hesitation. Each merged pull request then feeds richer examples back into the model, sharpening its contextual acuity and reinforcing the entire loop.

Most teams haven't entered the flywheel-yet

Only a small share of developers (just 3.8%) report experiencing both low hallucinations and high confidence in shipping AI-generated code. These are the teams truly benefiting from AI in production. They trust the suggestions, ship faster, and close the loop with high-quality feedback.

Among this low-hallucination group, those who also feel confident (17%) report:

- 1.3x higher likelihood of seeing code quality gains (44% vs. 35%)
- 2.5x greater confidence in shipping AI code (24% vs. 9%)

This is the group we think of as the "sweet spot" – and within it, over half (53%) report clear improvements in code quality.

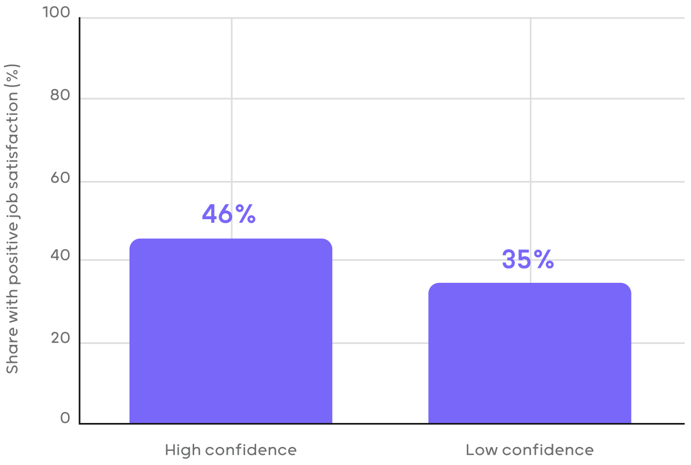
This suggests a strong link between accuracy, quality, and confidence. When developers see both fewer errors and higher - quality output, they're much more likely to trust the AI and use it in production.

We also see that low hallucinations make developers 1.3x more likely to say AI has improved code quality (44% vs. 35% overall). Still, most developers – even those with accurate output – remain hesitant. That's where automated quality checks can close the gap.

Confidence feels good: how trusting AI-generated code builds happier engineers

We asked developers how AI affects their job satisfaction–and found that confidence in the quality of the code matters just as much as the code itself.

Job-Satisfaction gain vs Confidence in AI code

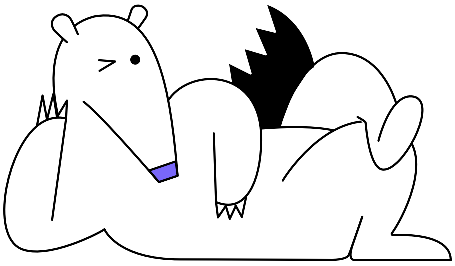


- Among developers who feel confident in AI-generated code, 46% say it makes their job more enjoyable.
- That drops to 35% among those who don't trust the output.

That 11-point gap isn't about raw velocity; it's about peace of mind. Confident teams spend less time second-guessing bots, chasing flaky tests, or re-writing low-quality diffs, freeing them to focus on creative problem-solving–the work that sparks joy and fuels retention.

Productivity powers happiness – confidence locks it in

Developer satisfaction rises sharply when AI boosts productivity. But even when productivity gains stall, confidence in the code still doubles the likelihood of a positive morale outcome.



The dual engines of satisfaction

Our data shows two key drivers of job satisfaction:

1. **Productivity** gains from AI
2. **Confidence** in AI-generated code

Positive job satisfaction (%) by confidence & productivity

	High confidence	Low Confidence
No improvement	25%	14%
Productivity	61%	58%

Developers who experience both – faster output and trust in results – report the highest rates of job satisfaction (61%). But even when only productivity improves, satisfaction remains high (58%), showing that speed alone carries weight.

However, when productivity doesn't improve, confidence becomes the swing factor.

Among developers who didn't see productivity gains:

- **25%** of those with high confidence still reported positive satisfaction
- Only **14%** of low-confidence developers felt the same

That's nearly a 2x difference, proving that trust in AI output can still move the needle on morale–even when performance stalls.

About Qodo

Qodo is an agentic code quality platform for reviewing, testing, and writing code, integrating AI across development workflows to strengthen code quality at every stage. Our AI agents are powered by a core platform that provides deep context awareness, enabling AI to understand the specific best practices of a codebase and solve complex coding challenges more effectively.

Plan → Write → Test → Review

Qodo Gen

IDE-Agent

Agentic code generation and development

- Autonomous coding agent with iterative workflows
- Extensible tooling via Model Context Protocol (MCP)
- Enterprise governance and control settings

Qodo Merge

Git-Agent

AI-powered code review agent

- 15 automated PR review workflows
- Intelligent code suggestions and issue detection
- Customizable review workflows and best practices

Codebase intelligence engine: Continuous codebase indexing, analysis and embedding for AI contextual awareness.

A system built to solve complex code with AI

With Qodo, AI doesn't run wild in your code. Our suite of AI agents are guided by quality-focused workflows, underpinned by a platform for context-awareness and codebase intelligence.

AI across the SDLC

Qodo supports the full SDLC with code reviews, test generation and coverage, agentic coding to deliver eliminate the speed vs. quality tradeoff.

Codebase intelligence engine

Continuously feed Qodo agents with your organization's best practices and context. Qodo's codebase intelligence engine semantically understands your entire codebase–it's structure, dependencies, and logic.

Multi-agent code integrity

Generate full-stack code, fix bugs, build UI components, and implement tasks from tickets. Automate full review workflows to surfacing issues, get suggestions, ensure compliance, and identifying missing tests.

Enterprise governance and control

Qodo provides smart guardrails that govern agent behavior, enforce rules, and ensure development workflows operate within a trusted set of tools.

Leverage top tier models

Gemini

Integrating with

Self-hosted, airgapped, VPC, Cloud

aws

intuit · intel · walkme · HiBob · melio · cisco · Walmart · Ford