Sonar

State of Code Developer Survey report

2026

Table of Contents

Introduction

Sonar analyzes over 750 billion lines of code each day, which gives us a unique understanding of code. This year, we kicked off a new report series called the State of Code to share some of our knowledge with developers and technology leaders more broadly.

We've already written reports on code reliability, security, maintainability, and the specific coding personalities of leading LLMs. Those reports focused primarily on the code itself and the models creating it. Next we wanted to expand that view to include the state of code from the perspective of the people doing the work—the developers writing code and collaborating with AI to build it.

Specifically, we wanted to get a read on what is changing for them. As AI rapidly shifts the mechanics of coding, we need to understand the on-the-ground reality—the efficiencies, the frustrations, and the new workflows emerging. To ensure we add real value to the industry narrative, we designed this study to build upon the findings in other leading developer surveys and to answer the pressing questions we still had after reading them.

After surveying more than 1,100 developers, we saw a critical new narrative emerging. Simply put, the explosion in AI-generated code hasn't led directly to massive and much-hyped productivity gains yet. Instead, a verification bottleneck has emerged, creating a whole new set of challenges. As we cover this and other findings, we'll explore how the AI-coding shift is manifesting across software engineering organizations across the world, and how they are adapting to address it.

About this report

The 2026 State of Code Developer Survey was a quantitative online survey conducted among professional software developers. Fieldwork for the survey ran throughout October 2025.

The final sample size for the study included 1,149 responses, distributed globally. All respondents were 18 years or older, employed full-time or self-employed in a technology role (the vast majority worked in software engineering, with some others in fields related to IT ops, data science, or product management), write code or manage developers using at least one programming language, and have used AI as part of their job within the past year.

Further details about the report's demographic makeup are available in the appendix.

How developers are (really) using AI

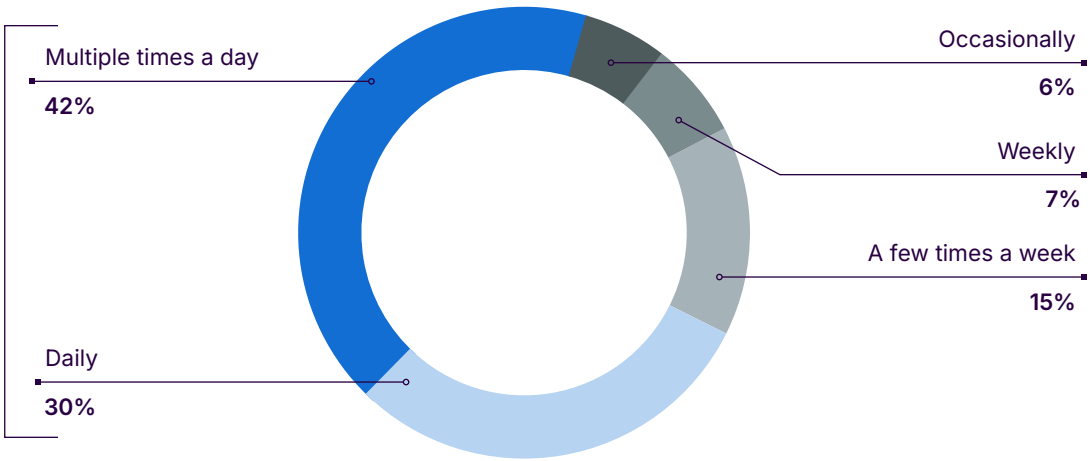
72% of developers who have tried AI use it every day

AI-assisted coding is officially a standard part of the developer workflow. 72% of developers who have tried AI coding tools now use them every day.

72% of developers who have tried AI use it every day

How frequently (do you / your team or company) use AI coding tools in your development workflow?

72%
every day

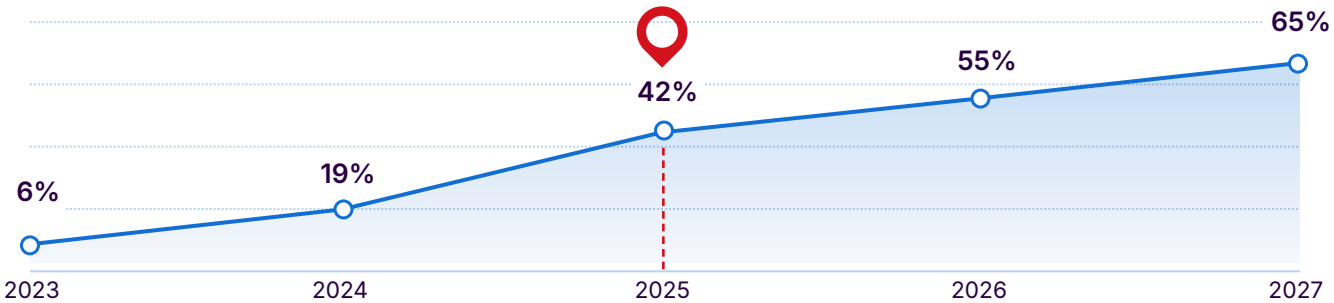


n=1,148

Developers also report that 42% of their code is currently AI-generated or assisted—a share that they predict will increase by over half by 2027, and up from only 6% in 2023.

Average share of AI-assisted or generated code committed by developers

What % of the code you committed or contributed was / will be generated or significantly assisted by AI tools?

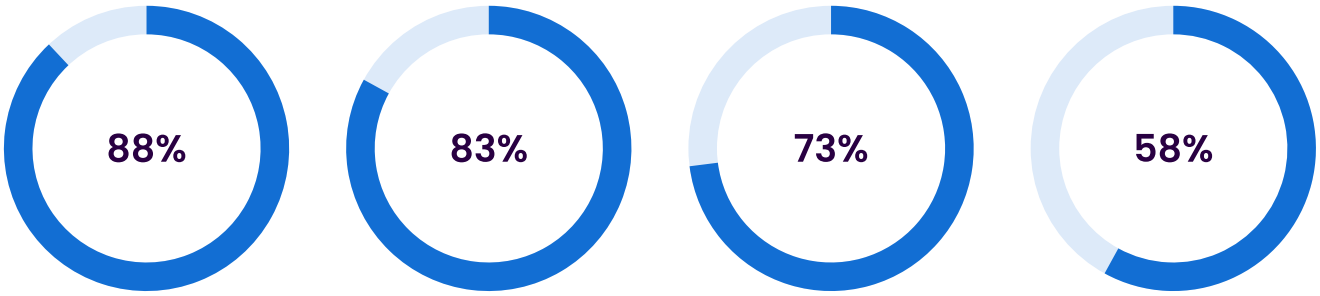


n=979

And AI is not just for side projects and experimentation. Developers are using AI across the gamut of software projects, from prototypes (88%) and internal, non-critical production software (83%) all the way to customer-facing applications (73%) and even mission-critical services (58%).

Developers are using AI across the gamut of software projects

Thinking about your team / company, which of the following types of work involves the use or assistance of AI?



Prototypes, experiments, proofs of concept

Production software for internal, non-critical workflows

Production software for customer-facing applications

Production software for business-critical or mission-critical services

n=1,149

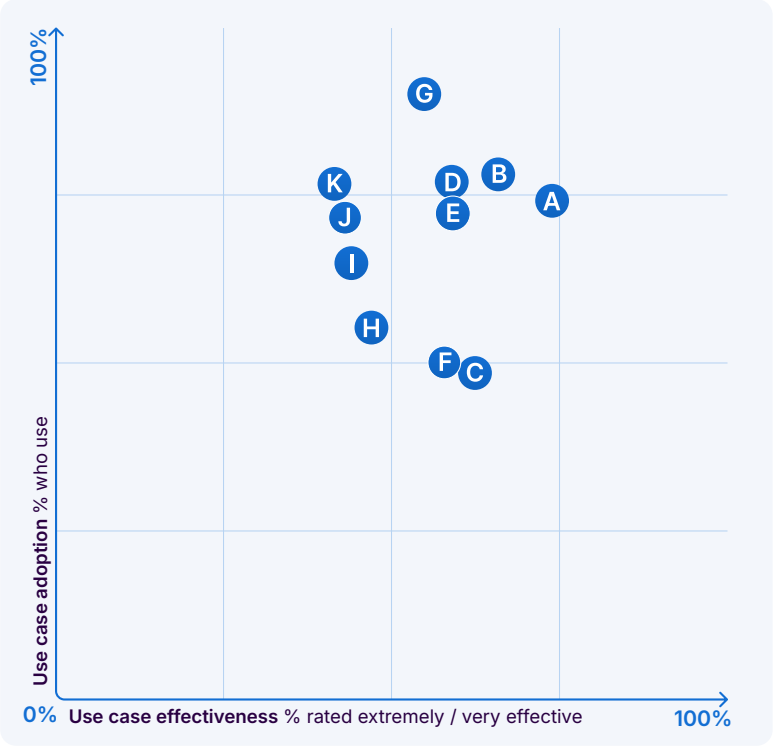
Use cases, and the gap between usage and effectiveness

Just because AI is used everywhere doesn't mean it's effective evenly. When we look at how developers are using AI versus how effective they find it for those specific tasks, a clear gap sometimes emerges. In a perfect world, adoption would increase more or less linearly with effectiveness. But in practice, we see use cases where developers have reported low effectiveness but higher rates of adoption.

Understanding AI use cases

For which of the following tasks is your team / company using AI coding tools?

How effective are AI coding tools for each of the following tasks you or your team / company has used them for?



	Use case	Effectiveness ↓	Adoption
A	Writing documentation	74%	74%
B	Explaining or understanding existing code	66%	78%
C	Vibe coding / creating new projects with mostly AI-generated code	62%	48%
D	Generating tests	59%	75%
E	Researching technical solutions or exploring APIs/libraries	59%	74%
F	Translating code from one language to another	58%	50%
G	Assisting development of new code	55%	90%
H	Code review	47%	55%
I	Debugging code	44%	65%
J	Refactoring or optimizing existing code	43%	72%
K	Adding or updating functionality in existing code	42%	76%

n=1,149

The best example of this is also the most common use case for AI: assisting with new code development (90% of developers). Only 55% of those users rated AI as "extremely or very effective" for that task.

Refactoring or optimizing existing code shows a similar effectiveness gap: while 72% of developers report using AI tools for this use case, only 43% attest to its effectiveness in that task.

Where AI really shines

The data shows AI tools are most effective at tasks that involve experimentation or working with what's already there.

The tasks where AI is most effective include:

- Writing documentation (74% effective)
- Explaining or understanding existing code (66% effective)
- Vibe coding / green-field prototyping (62% effective)
- Generating tests (59% effective)

Developers have embraced AI as a daily partner, but they're finding it's a much better "explainer" and "prototyper" than it is a "maintainer" or "refactorer"—at least for now. It's highly effective at generating new things (docs, tests, new projects) but struggles more with the complex, nuanced work of modifying and optimizing existing, mission-critical code.

The takeaway

Developers are pragmatic: they've fully embraced AI as a daily assistant, using it to write documentation and generate tests. But they also know its limits, showing less confidence in its ability to handle complex, existing code. This gap between high usage and selective effectiveness isn't just about features; it's about confidence. When the stakes are high, how much do developers really trust the code AI generates? This brings us to the core of the issue: developer trust.

Vibe check: Do developers trust AI?

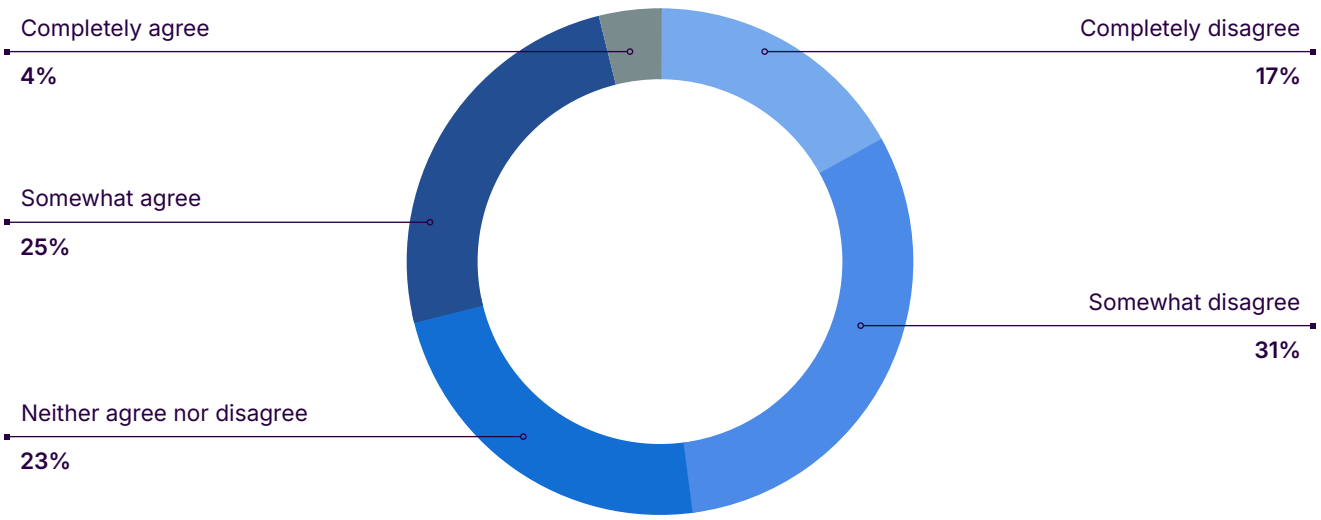
96% of developers don’t fully trust that AI-generated code is functionally correct

It's no secret that AI is changing development. Our study found that developers are seeing real benefits, reporting an average personal productivity boost of 35%. On top of that, more than half (54%) say they’re more satisfied with their job as a result of AI.

And while 82% of developers agree AI helps them code faster, and 71% say it helps solve complex problems more efficiently, this speed creates a new challenge: a trust gap. 96% of developers don’t fully trust that AI-generated code is functionally correct.

96% of developers don’t fully trust that AI-generated code is functionally correct

To what extent do you agree with each of the following statements as it relates to the use of AI coding tools and AI-assisted or generated code? | Choice: I trust that AI code is functionally correct



n=1,149

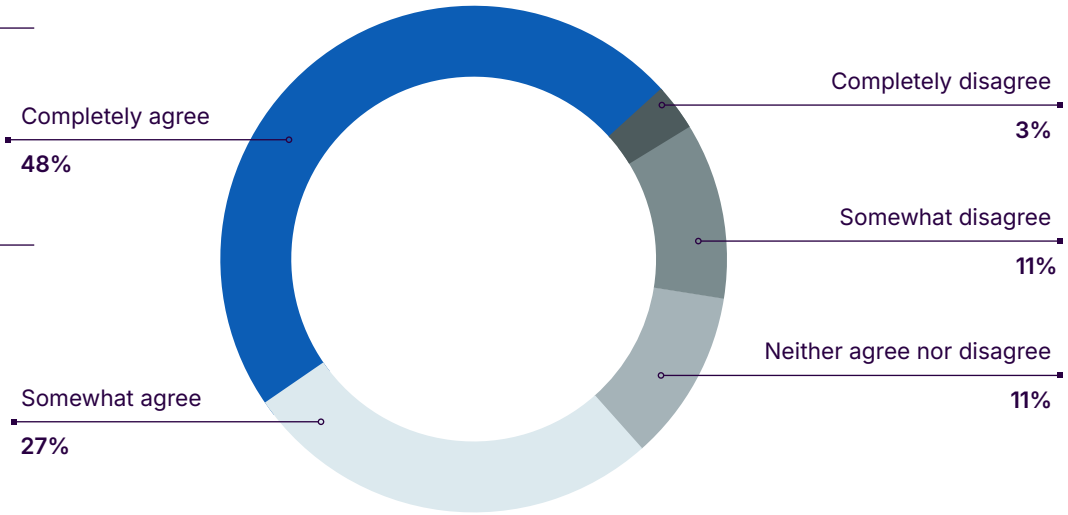
The verification bottleneck

Given our finding in the prior section that 96% of developers have a hard time trusting that AI-generated code is functionally correct, you would think that verification of AI code is widespread. However, this is not the case: only 48% of developers always check their AI-assisted code before committing.

48% of developers always check their AI-assisted code before committing

To what extent do you agree with each of the following statements as it relates to the use of AI coding tools and AI-assisted or generated code? | Choice: I always check my AI-generated or assisted code before committing it

48%
completely agree



n=1,149

This verification step isn't trivial. While AI is supposed to save time, developers are spending a significant portion of that saved time on review. Nearly all developers (95%) spend at least some effort reviewing, testing, and correcting AI output. A majority (59%) rate that effort as "moderate" or "substantial."

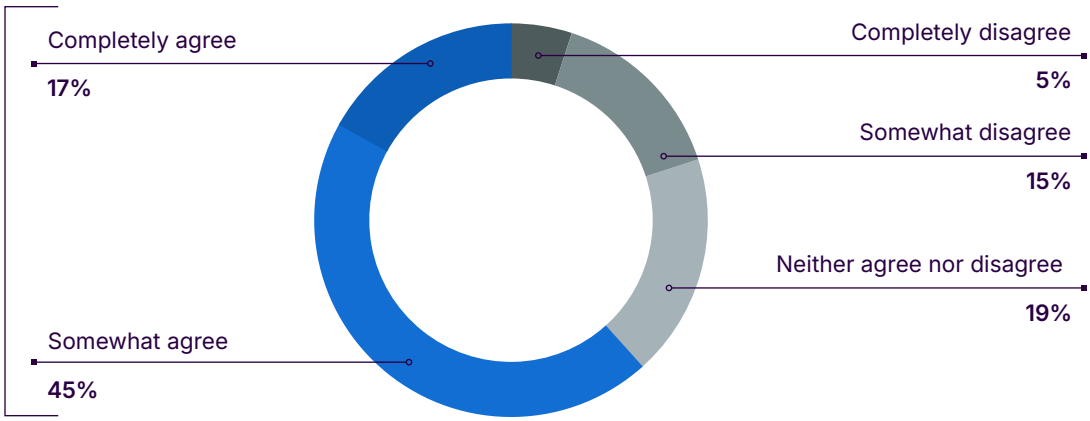
In fact, 38% of developers say reviewing AI-generated code requires more effort than reviewing code written by their human colleagues. (This contrasts with only 27% who say it requires less effort.)

Why is it so much work? 61% agree that "AI often produces code that looks correct but isn't reliable." That's a critical finding—it means AI code can introduce subtle bugs that are harder to spot than typical human errors. The same percentage (61%) agree that it "requires a lot of effort to get good code from AI" through prompting and fixing.

61% of developers agree that AI often produces code that looks correct but isn't reliable

To what extent do you agree with each of the following statements as it relates to the use of AI coding tools and AI-assisted or generated code? | Choice: AI often produces code that looks correct but isn't reliable

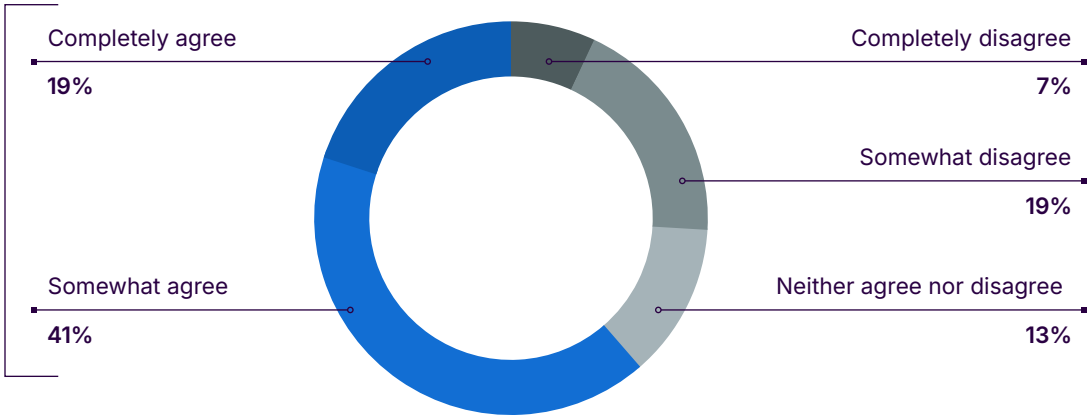
61%
agree



61% of developers agree that it requires a lot of effort to get good code from AI

To what extent do you agree with each of the following statements as it relates to the use of AI coding tools and AI-assisted or generated code? | Choice: It requires a lot of effort (in prompting, fixing, etc.) to get good code from AI

61%
agree



n=1,149; Percentages are rounded

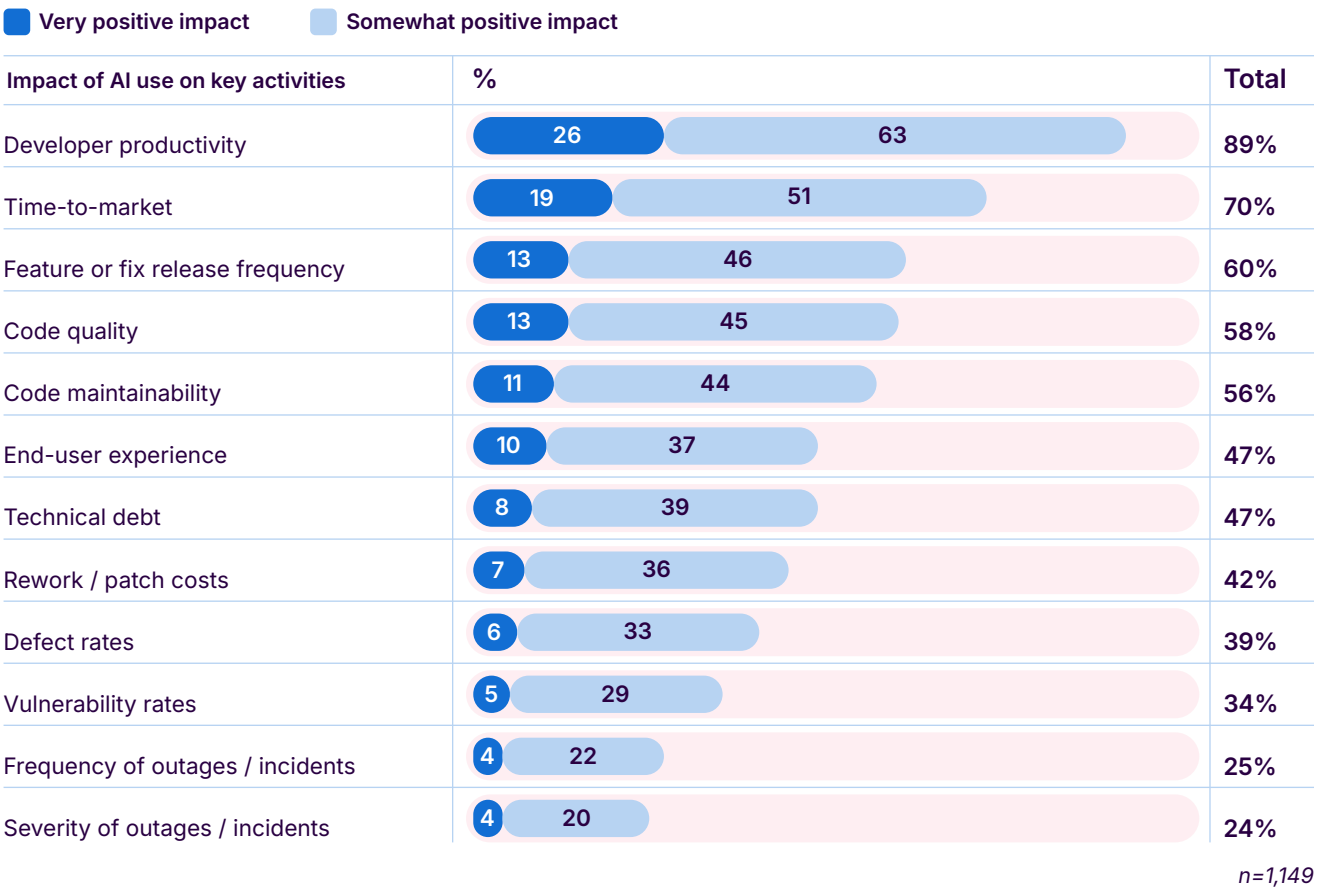
Where AI's impact is felt (and where it's not)

Developers are clearly shipping code faster, but whether this consistently translates to better outcomes is less clear.

While 70% say AI has positively impacted their time-to-market, less than half (47%) say it's had a positive impact on the end-user experience or on reducing technical debt. This makes sense: if you're shipping code that looks right but isn't reliable, you're not improving the user's experience or the long-term health of your codebase.

AI impact is mostly seen in developer productivity and time-to-market, but there's room for growth in multiple other aspects

What impact has AI-generated or assisted code had on your team / company for each of the following?



Impact of AI use on key activities	%	Total
Developer productivity	26 63	89%
Time-to-market	19 51	70%
Feature or fix release frequency	13 46	60%
Code quality	13 45	58%
Code maintainability	11 44	56%
End-user experience	10 37	47%
Technical debt	8 39	47%
Rework / patch costs	7 36	42%
Defect rates	6 33	39%
Vulnerability rates	5 29	34%
Frequency of outages / incidents	4 22	25%
Severity of outages / incidents	4 20	24%

n=1,149

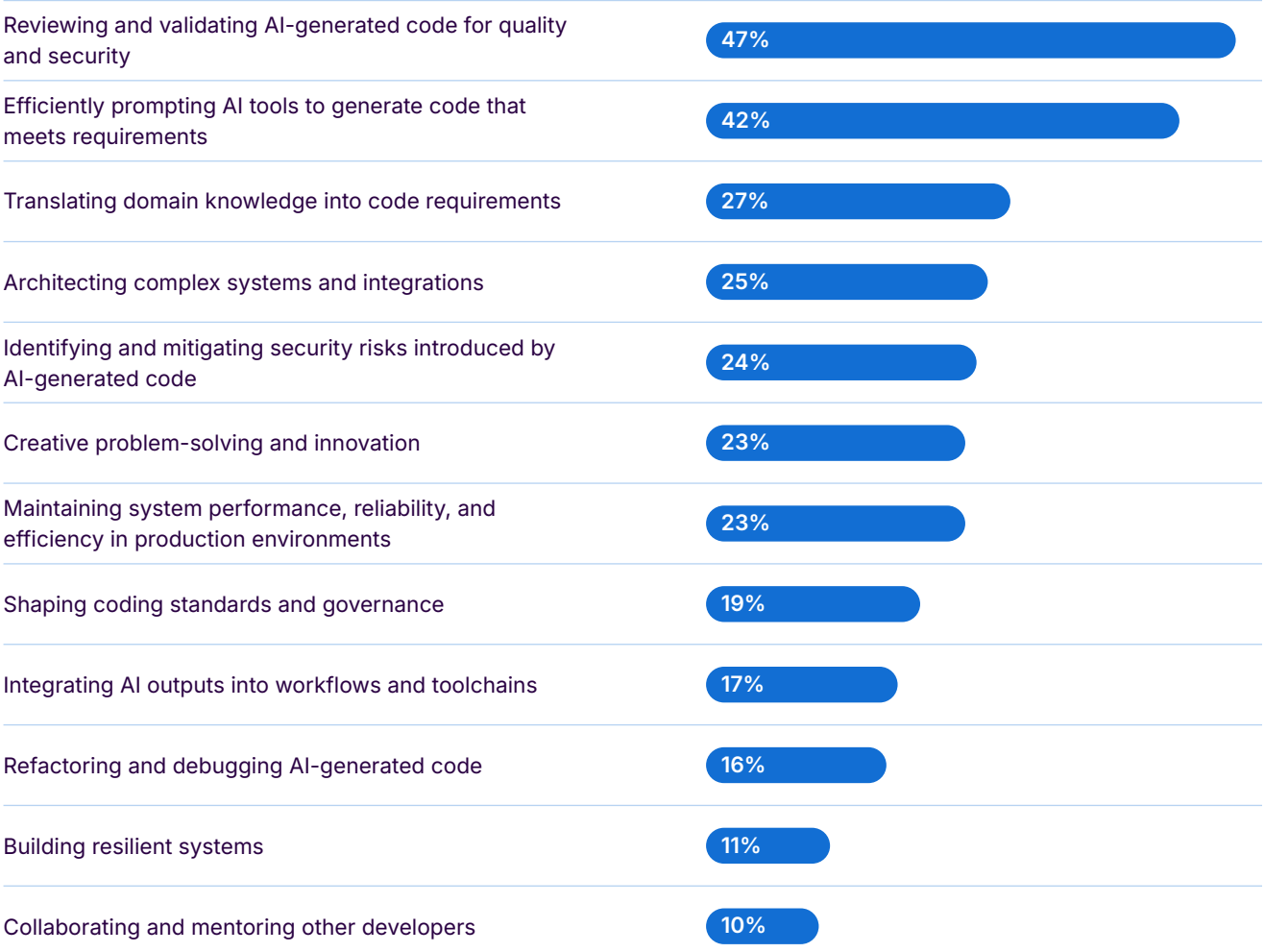
SonarQube users report stronger positive impacts on code quality, technical debt, rework costs, defects, and vulnerabilities than non-users. This suggests that having a systematic verification process in place is key to turning AI's speed into real-world quality improvements.

A new set of skills and concerns

This new "verify" step is also redefining what it means to be a developer. When we asked what skills will be most important in the AI era, the number one answer was "reviewing and validating AI-generated code for quality and security" (47%). This was followed by "efficiently prompting AI tools" (42%).

Code quality review & validation is the most important skill for developers in the AI era

Which of the following skills will be most important for developers to have in the evolving AI-assisted or generated coding environment?



n=1,149

Ultimately, AI is speeding up code generation, but it's also created a bottleneck at the verification stage of software development, with more work now required to review code.

The takeaway

It's clear that this new verification bottleneck is the central challenge in the age of AI-assisted coding. But one critical component of code review is knowing the provenance of the code. This raises a critical question: which code generation tools are even being used, and how are developers accessing them? This leads us to another key finding: the rapid, often ungoverned, sprawl of AI tools.

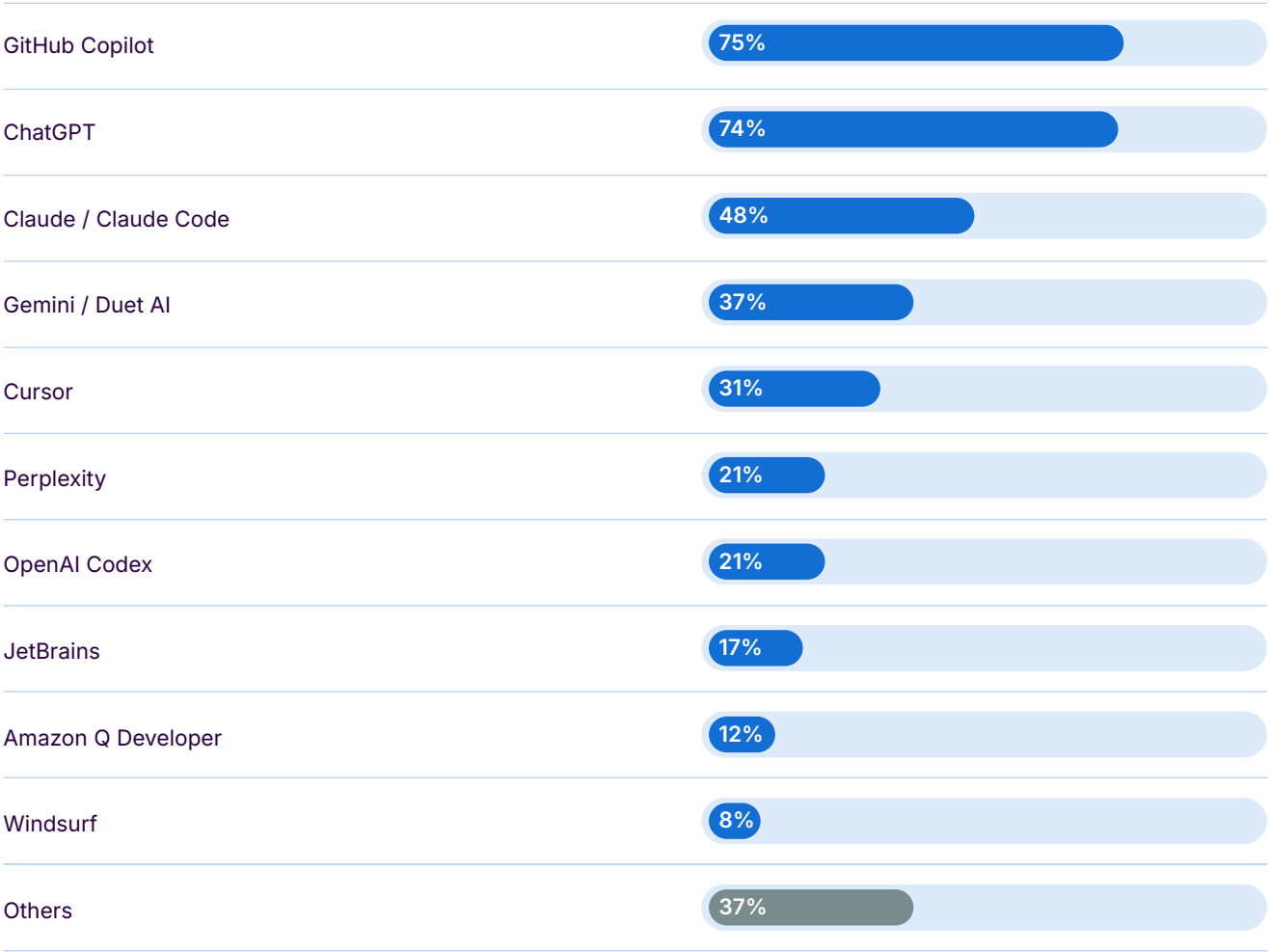
The top AI tools, and how they're used

Copilot and ChatGPT lead the field

AI adoption isn't just happening; it's already starting to coalesce around favorites. The two most dominant tools, GitHub Copilot and ChatGPT, are used by 75% and 74% of developers, respectively, with Claude following at 48%.

GitHub Copilot and ChatGPT are the most widely used tools for software development tasks

In the past year, which of the following AI coding tools or AI features have you or your team / company used for software development tasks?

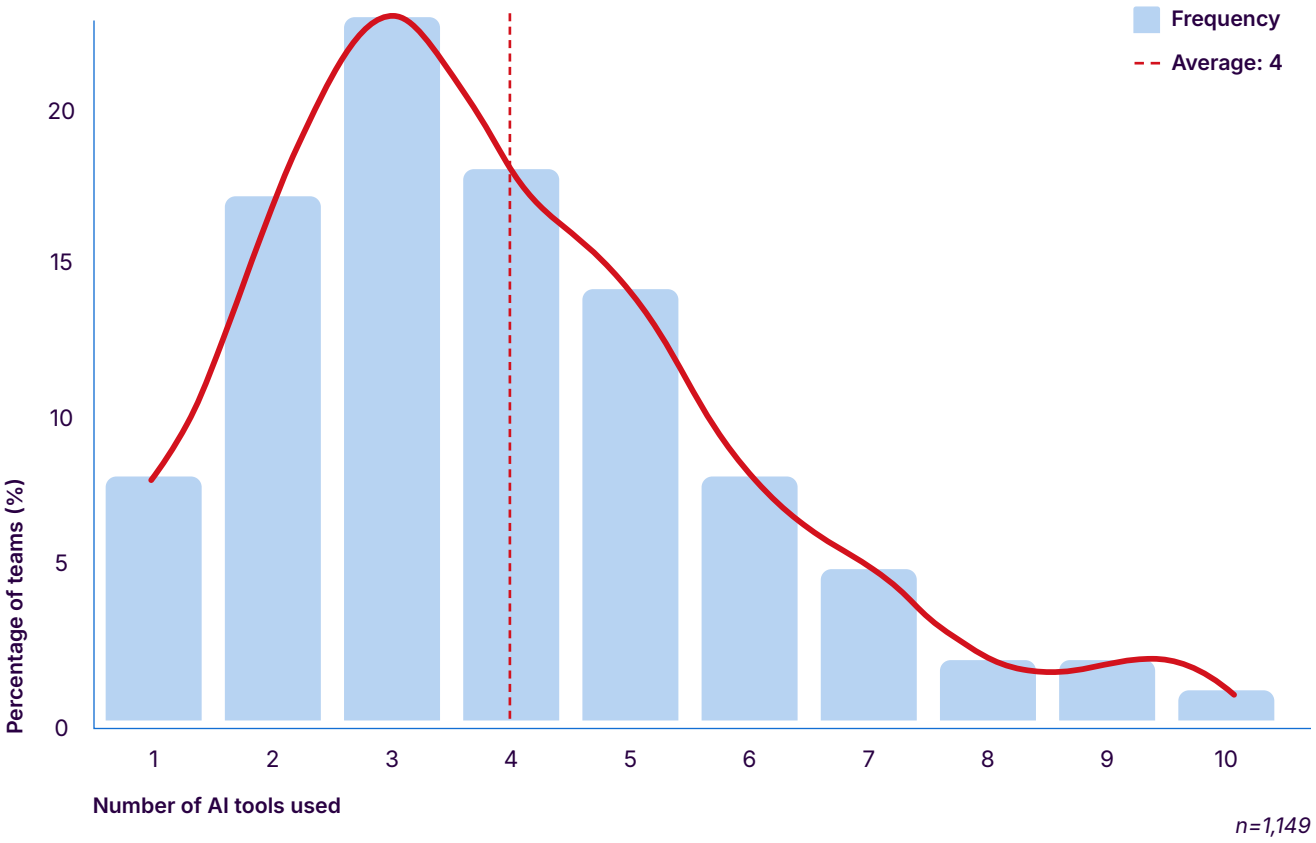


n=1,149

But, the data reveals a complex and fragmented picture: on average, development teams juggle four different AI tools.

The average development team uses four different AI tools

Based on aggregated answers to the following question: In the past year, which of the following AI coding tools or AI features have you or your team / company used for software development tasks?



n=1,149

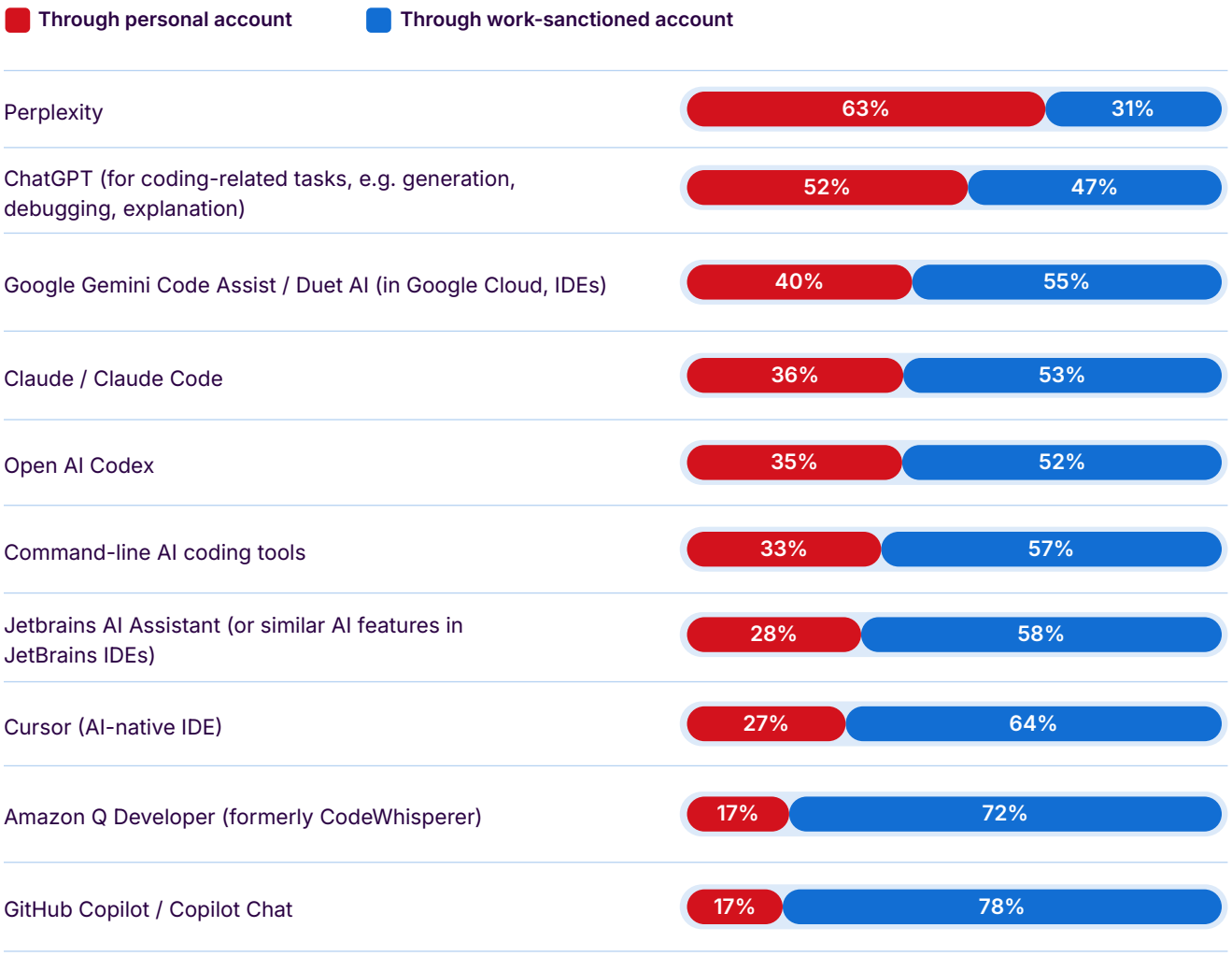
For engineering leaders, this means that while AI is boosting productivity, it's also introducing a "bring your own AI" (BYOAI) culture that's running ahead of official governance.

The personal account problem

It also turns out that many of the tools developers are using aren't fully approved by their workplaces. Across the top 10 AI tools, our data shows that 35% of developers are accessing them through personal accounts rather than work-sanctioned ones.

Over 50% of developers use ChatGPT through personal accounts while 78% use GitHub Copilot through work accounts

In the past year, how have you personally used each of the following AI coding tools or AI features for software development tasks as part of your work?



n=1,147; bases vary

ChatGPT is a perfect example. While 74% of developers have used it in the past year, 52% of those users are accessing it via their personal accounts. This trend is even more pronounced with tools like Perplexity, where 63% of its users are logging in via personal accounts.

This shadow adoption creates a massive blind spot for security and compliance. When developers use personal accounts, they might be feeding sensitive company or customer data into public models, creating serious risks.

Interestingly, this isn't a universal problem. Some tools are clearly being adopted through official channels. GitHub Copilot and Amazon Q Developer, for example, show much lower personal account use (17% for both), suggesting a more formal, top-down rollout. And Cursor is similar, showing only 27% in personal account use compared to 64% on work-sanctioned accounts.

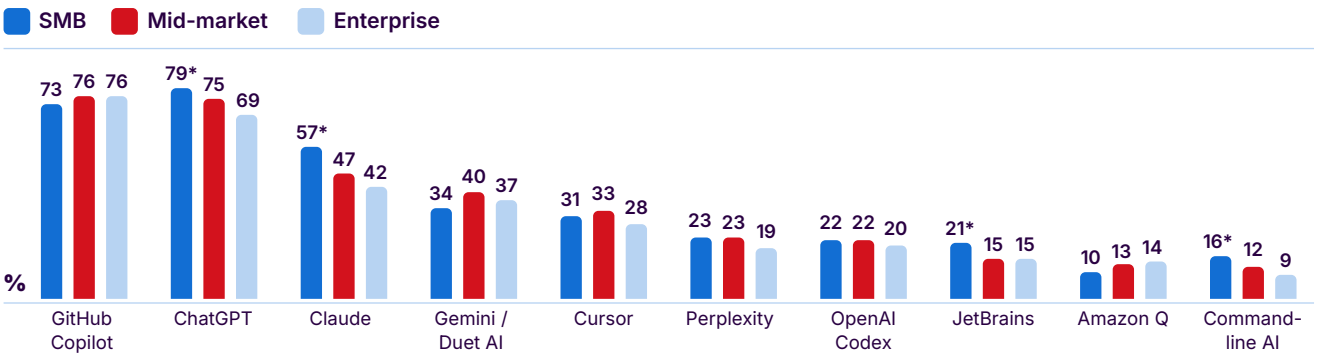
Who is using which tools?

The data also shows that tool choice varies based on company size and developer experience.

- Company size: Smaller companies (SMBs) are more likely to be using ChatGPT, Claude, JetBrains, and command-line AI tools than their larger counterparts.

ChatGPT, Claude and JetBrains are favorites among SMBs

In the past year, which of the following AI coding tools or AI features have you or your team / company used for software development tasks?

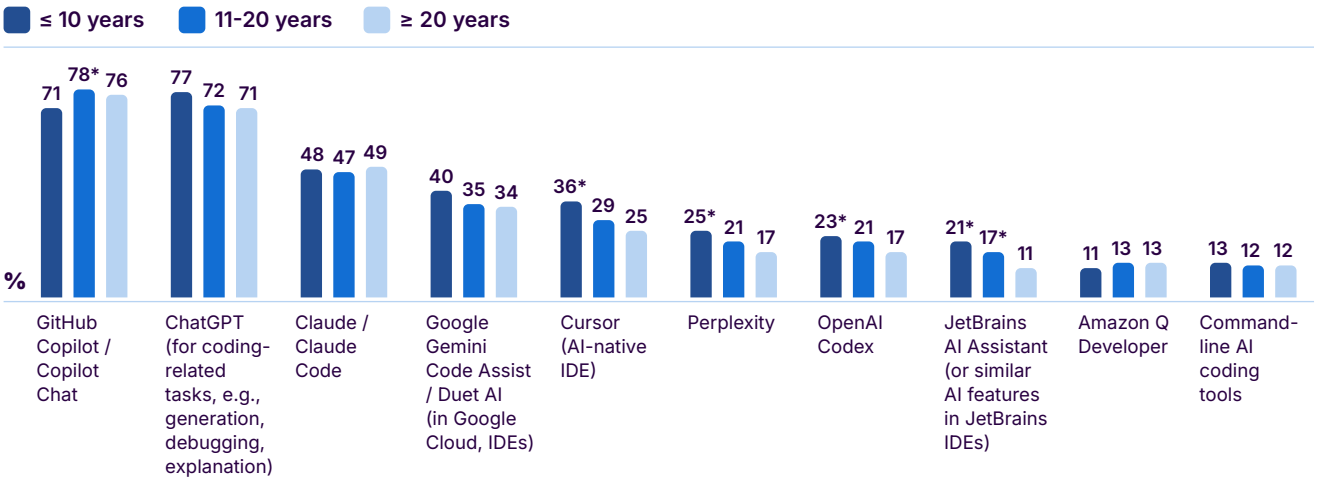


*n=1,149; *indicates statistical significance*

- Experience level: Junior developers are more likely to be using a more varied set of tools, including Cursor, Perplexity, OpenAI Codex, and JetBrains AI.

Cursor, Perplexity, OpenAI Codex are favored by junior developers

In the past year, which of the following AI coding tools or AI features have you or your team / company used for software development tasks?



*n=1,149; *indicates statistical significance*

This data paints a clear picture: developers aren't waiting for permission. They are actively experimenting with a wide array of tools to get their work done, often on their own personal accounts. For engineering leaders, the challenge isn't if AI will be used, but how to manage the new risks it introduces without slowing down the clear productivity gains.

The takeaway

This fragmented, "bring your own AI" culture highlights the first major wave of AI adoption: developers are seizing on assistant tools to speed up individual tasks. But this is just the beginning. As developers look to move beyond simple code generation to full task automation, they're turning to agentic AI. We'll explore this next, looking at how developers are beginning to use these autonomous agents and what it means for the future of the development lifecycle.

The second act of AI: Agents

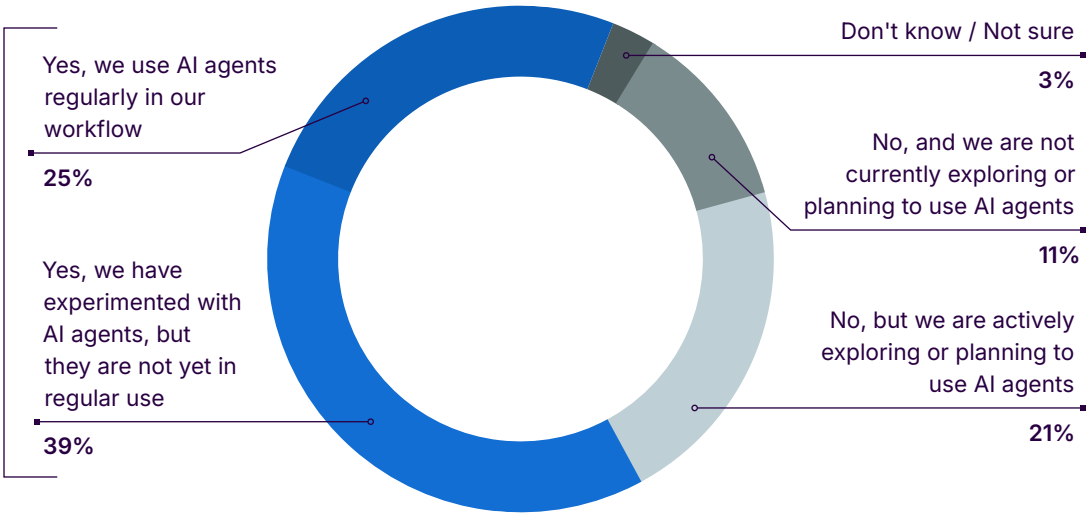
64% of developers have started to use AI agents

A key finding from this study is that agentic AI is moving from experiment to everyday tool. 25% of developers report using agentic AI tools regularly in their workflows, and another 39% have experimented with them. This means a combined 64% of developers have already started using these advanced agents.

64% of developers have started to use AI agents

Is your team / company currently using, or have you experimented with, AI agents in your software development work?

64%
using AI agents



n=1,149

Agentic use cases match AI's natural strengths

The data on agentic AI usage shows a strong focus largely on tasks where we know AI naturally performs reasonably well. Among the 64% of developers using agentic AI, here are the top use cases:

- Creating code documentation (68% of developers)
- Automated test generation and execution (61% of developers)
- Automated code review (57% of developers)

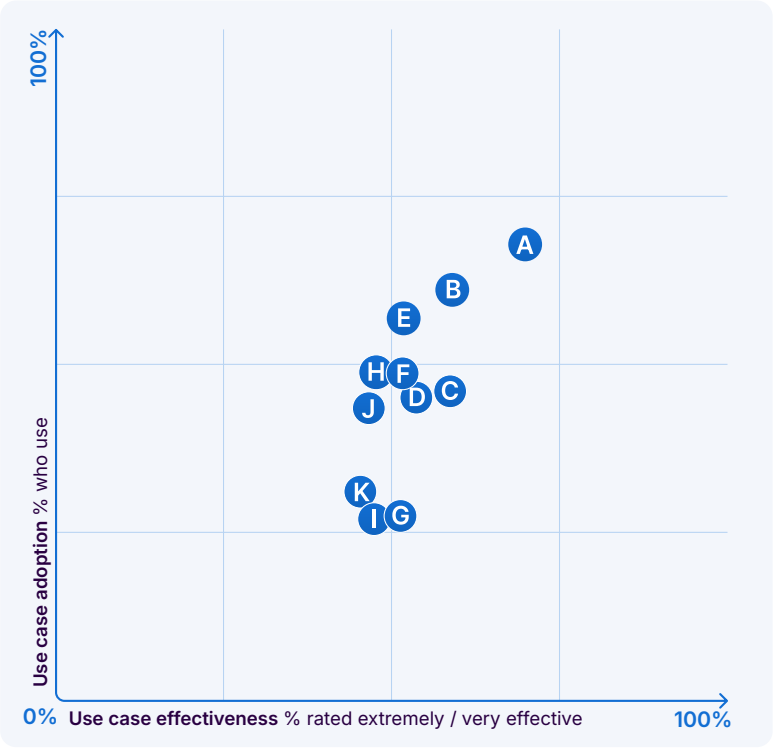
On the other end of the spectrum, the least common use case is security vulnerability patching or remediation, with only 28% of developers using agents for this task.

These popular uses line up well with their perceived effectiveness. 70% of developers rated creating code documentation as a somewhat or very effective agentic AI task. 59% of developers rated automated test generation as somewhat / very effective, and automated code review received 52%, suggesting that developers are finding real value in the tasks they're automating most often.

Understanding agentic AI use cases

For which of the following tasks is your team / company using AI agents?

How effective are AI agents for each of the following tasks you've used them for?



	Use case	Effectiveness ↓	Adoption
A	Creating code documentation	70%	68%
B	Automated test generation and execution	59%	61%
C	Vibe coding apps that create code based on conversational language	59%	46%
D	Project planning, task breakdown, or requirements analysis	54%	45%
E	Automated code review	52%	57%
F	Automated code generation for entire features or modules	51%	48%
G	Automated infrastructure setup or management	51%	29%
H	Automated code refactoring, modernization, or optimization	49%	48%
I	Security vulnerability patching or remediation	48%	28%
J	Automated debugging and issue remediation	47%	44%
K	Automated deployment pipeline configuration or management	47%	33%

n=737

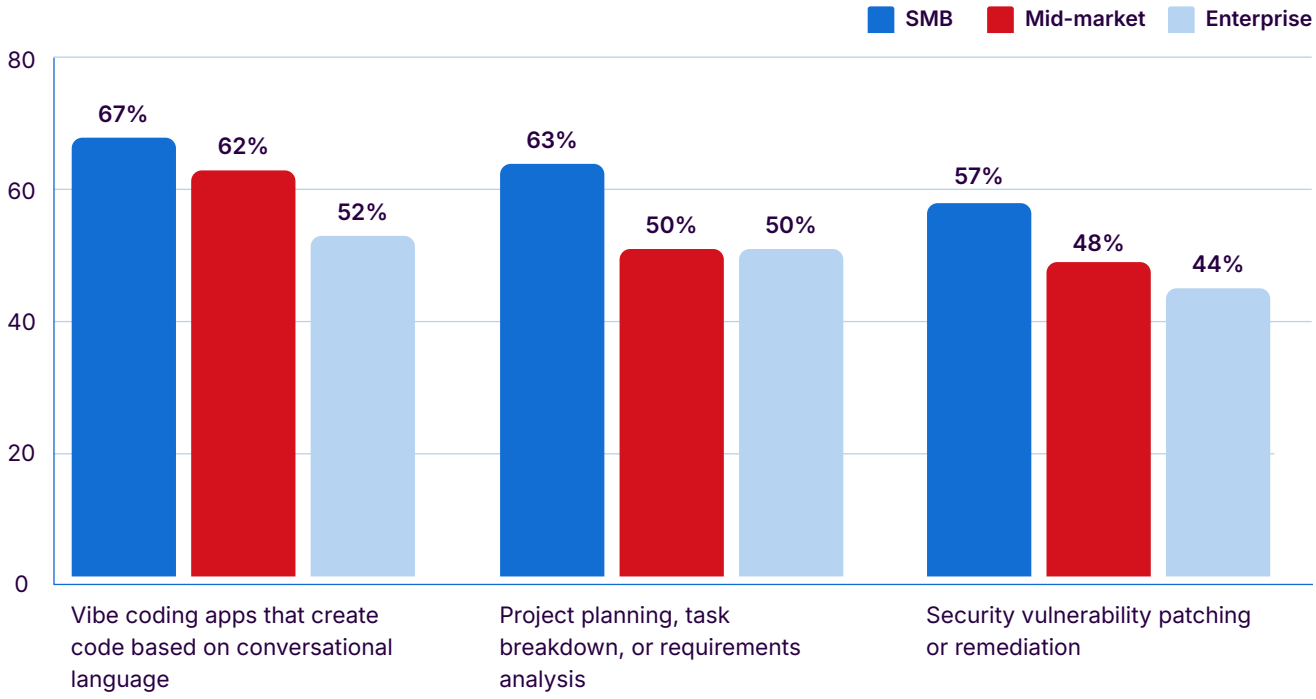
Where different teams see success

While documentation is the clear all-around winner, the data reveals some fascinating, specialized bright spots for different groups.

For example, developers at small-to-medium businesses (SMBs) are finding success with generative tasks. 67% of developers at SMBs reported that AI agents were effective for vibe coding tasks (using conversational language to create apps). This is significantly higher than peers at enterprise organizations (52%), suggesting agents are a powerful force multiplier for smaller, agile teams. Other bright spots for SMBs include project planning, task breakdown, or requirements analysis (63% of developers rated AI as effective) and, interestingly, security vulnerability patching or remediation (57%).

Use cases where AI agents offer more value to SMBs

How effective are AI agents for each of the following tasks you've used them for?

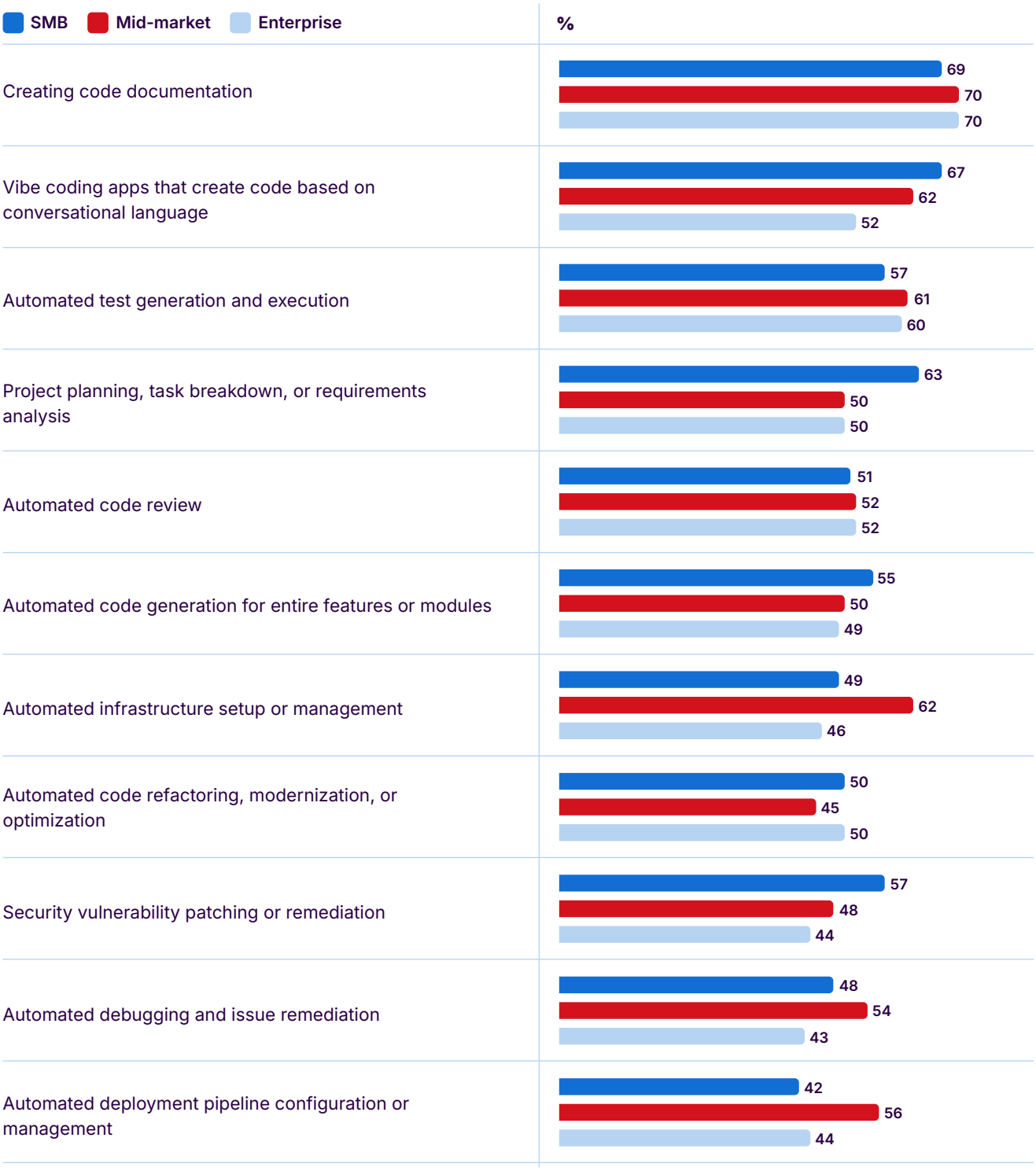


n=737

In between SMBs and enterprises, we find that developers at midsized organizations are also seeing success in key areas related to infrastructure setup. In particular, more developers report effectiveness for use cases like automated infrastructure setup or management (62%) and automated deployment pipeline configuration or management (56%).

AI agent use cases that work (and don't work) for different company sizes

How effective are AI agents for each of the following tasks you've used them for?



n=737

The takeaway

It's clear that agentic AI is beginning to successfully automate key parts of the development process, with a quarter of developers already using it regularly for tasks like documentation and test generation. But does automating these specific jobs mean developers are finally free from frustrating, repetitive work? Despite automating these jobs, AI might not be eliminating toil, but simply changing its shape.

Meet the new developer toil

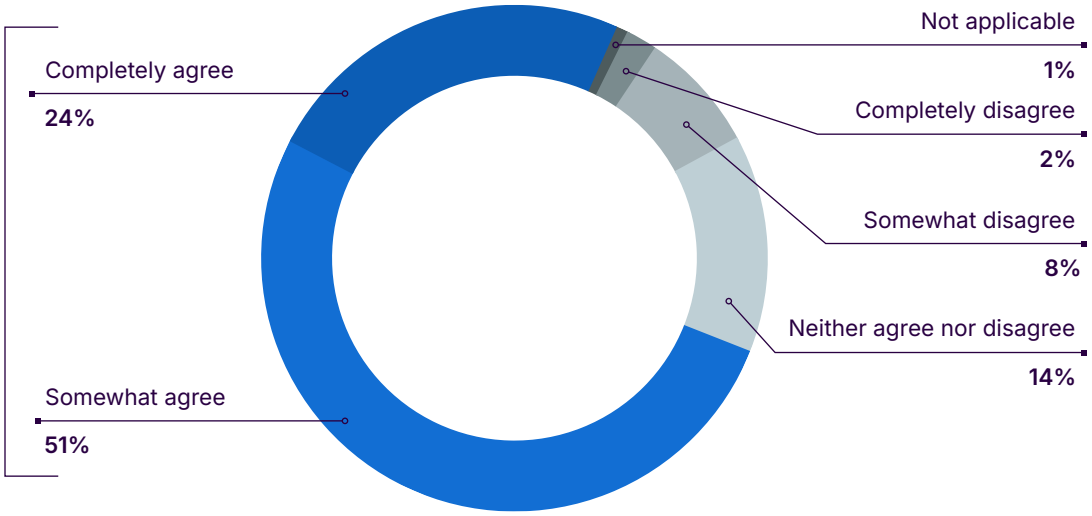
The illusion of toil savings

As mentioned in the “Vibe check: Do developers trust AI?” chapter, developers are seeing higher levels of productivity when using AI coding tools. They also reported toil reduction: 75% of developers said that AI reduces their “toil work”—tasks that hinder developer productivity or increase frustration.

75% of developers believe that AI reduces amount of time spent on toil work

To what extent do you agree with each of the following statements as it relates to the use of AI coding tools and AI-assisted or generated code? | Choice: AI reduces the amount of time I spend on toil work

75%
agree



n=1,149

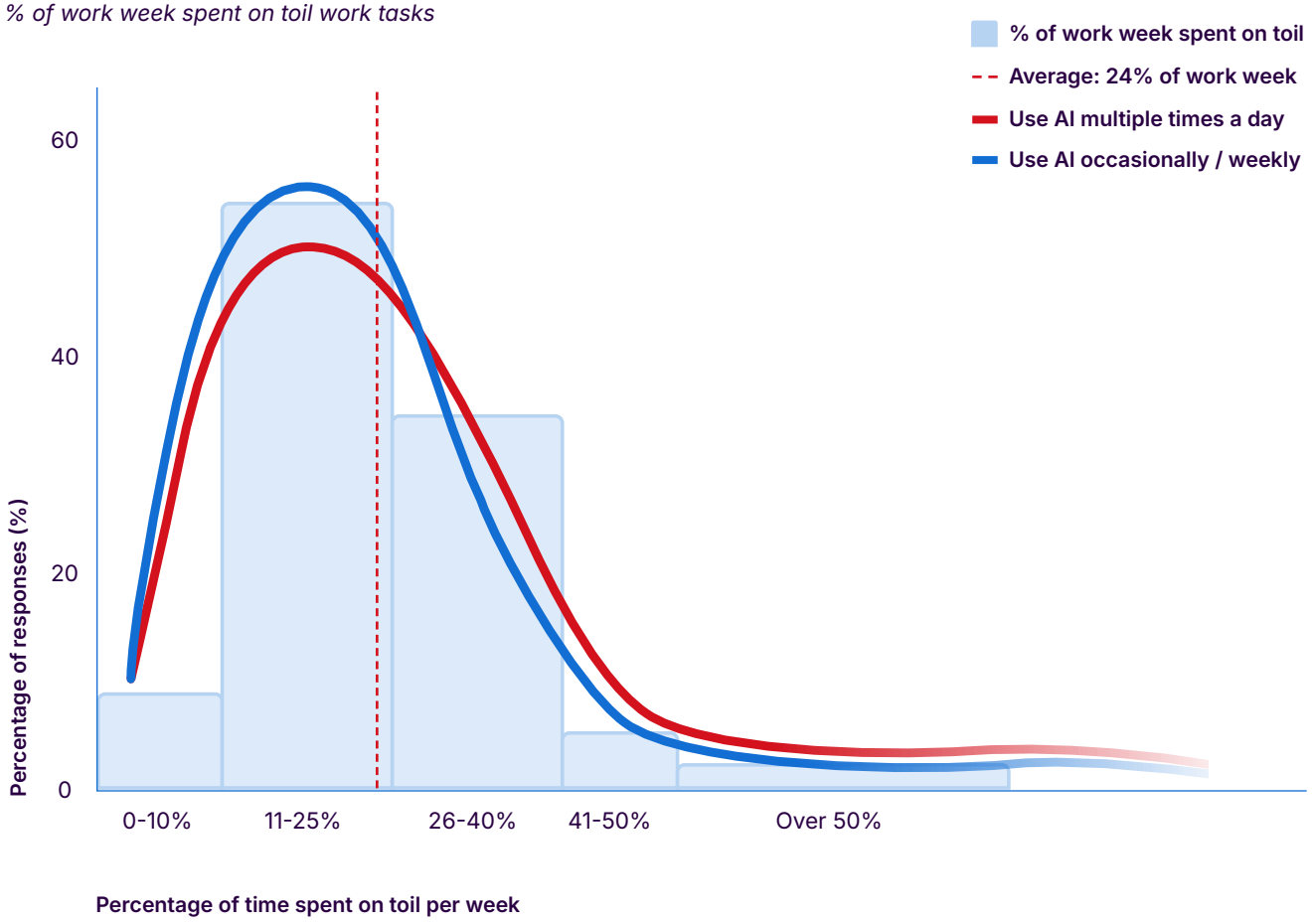
But under the surface, the picture becomes more complicated.

When asked about the portion of their work week assigned to tasks associated with toil (e.g. managing technical debt, debugging legacy or poorly documented code), developers reported spending nearly a quarter of their time on toil work.

Interestingly, the amount of time spent on toil (an average of 23-25%) stays almost exactly the same for developers who use AI coding tools frequently and for those who use them less often.

Time spent on toil

% of work week spent on toil work tasks



n=1,149

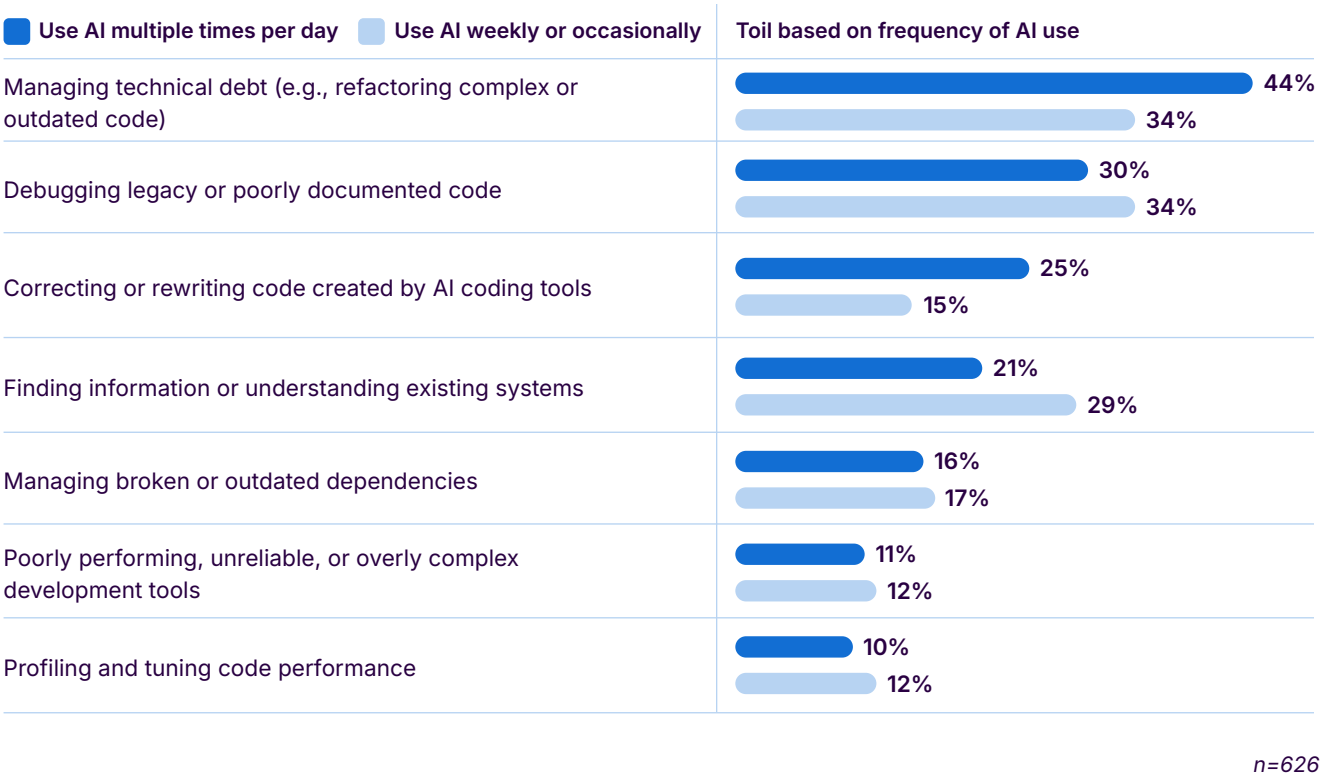
The great toil shift

It turns out that while developers spend less time on certain toil tasks, AI is not eliminating toil entirely; it is simply shifting its nature. Developers are swapping older frustrations for new ones:

- Less frequent AI users are more likely to report toil from tasks like debugging poorly documented code and understanding existing systems. These are exactly the kinds of problems that AI coding tools are good at solving.
- The most frequent AI users, however, are more likely to see toil in new areas: managing technical debt and, unsurprisingly, correcting or rewriting code created by AI coding tools.

Toil didn't shrink with AI, it changed flavor: more frequent users of AI are more likely to report toil in different areas

Thinking about your current workflows, what are your biggest sources of "toil work" that hinder / sap productivity or increase frustration in your current role?



n=626

This data suggests that while AI helps clear away old development hurdles, it creates new ones. We've accelerated code generation, but that just moves the pressure downstream to code management and verification.

Teams are doubling down on deterministic review

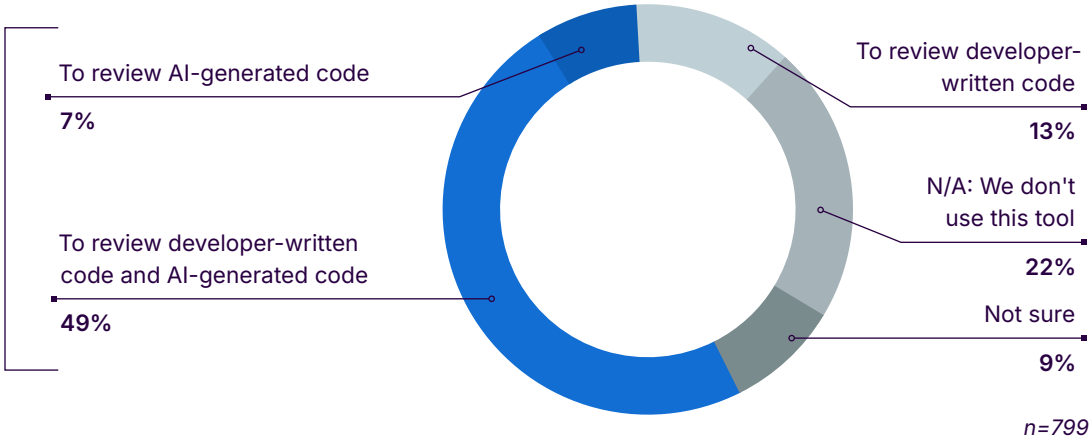
Developers are already adapting to this new reality. The data shows a strong reliance on static analysis tools, which are deterministic code review solutions, to manage the surge of new code from AI.

- Static analysis tools are widely adopted, with 70% of developers using them.
- 57% are already applying them to review AI-generated code.

How developers use static analysis code review

How is your company or team currently using the following automated code review tool? | Choice: Static Analysis Tool

57%
use static analysis to review AI code



n=799

There is also a pronounced swing towards applying static analysis review to AI code among enterprise developers: 60% of them review AI code with static analysis, compared to 51% of SMB developers. This is likely a side effect of higher investments by enterprises into processes to protect code quality across their codebases.

This isn't a temporary trend. Looking ahead, developers expect their reliance on these tools to increase. The perceived value of deterministic, rules-based code review is set to grow from 60% today to 68% over the next two years.

The takeaway

While AI is successfully changing how developers work, it hasn't reduced the overall burden of toil. It has simply focused that toil on a new, critical skill: verification. As a result, the industry is reinforcing its commitment to trusted, automated analysis to ensure all code—whether written by developer or an AI—is secure, reliable, and production-ready.

And this new verification challenge isn't just about managing messy code or technical debt; it's about confronting a much higher-stakes issue: the security of their code. This complex relationship between AI-generated code and code security is exactly what we'll explore next.

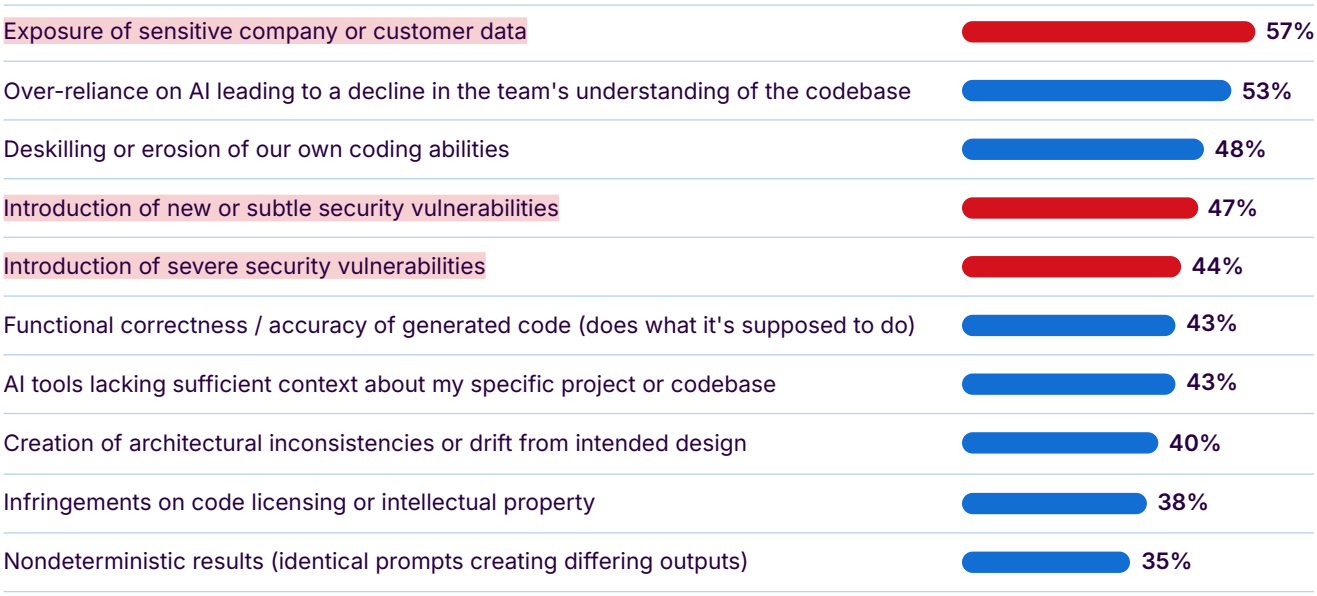
The tricky relationship between AI and code security

57% of developers worry that using AI risks sensitive data exposure

According to our study, the biggest concern developers have about AI code generation is security—specifically, 57% of developers worry about the risk of AI code exposing sensitive company or customer data. This isn't a minor issue; it's a majority of developers voicing a critical fear about the tools they're increasingly required to use.

Top 10 concerns developers have about AI-assisted or generated code

How concerned are you about each of the following when it comes to AI-assisted or generated code today? | Choice: Extremely / Very Concerned



n=1,149

This anxiety doesn't stop at data leaks. Developers are also on high alert for what the AI is creating. Nearly half (47%) are concerned about the introduction of new or subtle security vulnerabilities, and 44% worry about AI introducing severe vulnerabilities. It's clear that while AI boosts speed, it's also creating a new layer of risk that developers are now responsible for managing.

Enterprises feel the risk most

These AI concerns are most acute in enterprise environments. In large enterprises (over 1,000 employees), the percentage of developers concerned about exposure of sensitive company or customer data jumps to 61%. These organizations, which live and die by data integrity and compliance, are understandably the most cautious.

In fact, enterprises are significantly more concerned than smaller companies about:

- Direct prompt injections (34%, compared to 25% for SMB)
- Indirect prompt injections (35%, compared to 25% for SMB)
- Ensuring compliance with industry and organizational standards (38%, compared to 28% for SMB)

AI concerns: Enterprises vs. SMBs

How concerned are you about each of the following when it comes to AI-assisted or generated code today?

	SMB	Enterprise
Exposure of sensitive company or customer data	54%	61%*
Over-reliance on AI leading to a decline in the team's understanding of the codebase	56%	53%
Deskilling or erosion of our own coding abilities	49%	47%
Introduction of new or subtle security vulnerabilities	44%	49%
Introduction of severe security vulnerabilities	46%	45%
Functional correctness / accuracy of generated code (does what it's supposed to do)	44%	43%
AI tools lacking sufficient context about my specific project or codebase	43%	43%
Creation of architectural inconsistencies or drift from intended design	42%	38%
Infringements on code licensing or intellectual property	36%	39%
Nondeterministic results (identical prompts creating differing outputs)	29%	39%*
Performance and reliability of generated code	38%	32%
Ensuring compliance with industry-specific or organizational coding standards	28%	38%*
Direct prompt injections	25%	34%*
Indirect prompt injections	25%	35%*

*n=837; *indicates statistical significance*

A traditional risk management approach

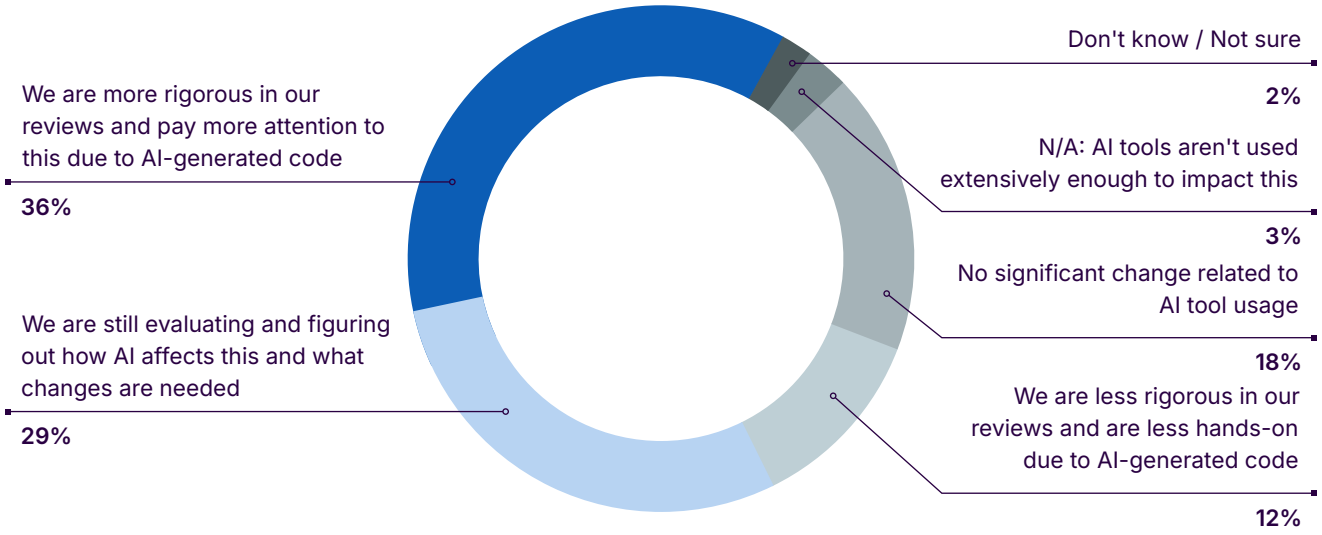
What does this mean for developers? Our data shows that they're clearly worried about security, but they aren't yet comfortable leveraging AI to solve it.

One of the use cases commonly touted for agentic AI tools is fixing security issues in code. But our research shows that while security is a top concern, it's the least common use case for agentic AI: only 28% of developers are using AI agents for security vulnerability patching or remediation. There's a major gap between what developers need (secure code) and what they're using AI for.

So how are organizations responding to this new risk profile? The truth is that at the moment, there's no clear consensus. Roughly a third of development teams report they are now more rigorous about code security, code quality, and compliance as a direct result of AI's growing influence. But for each of those, roughly another third of respondents report that they are still evaluating what changes will be needed.

36% of organizations are more rigorous about code quality because of AI

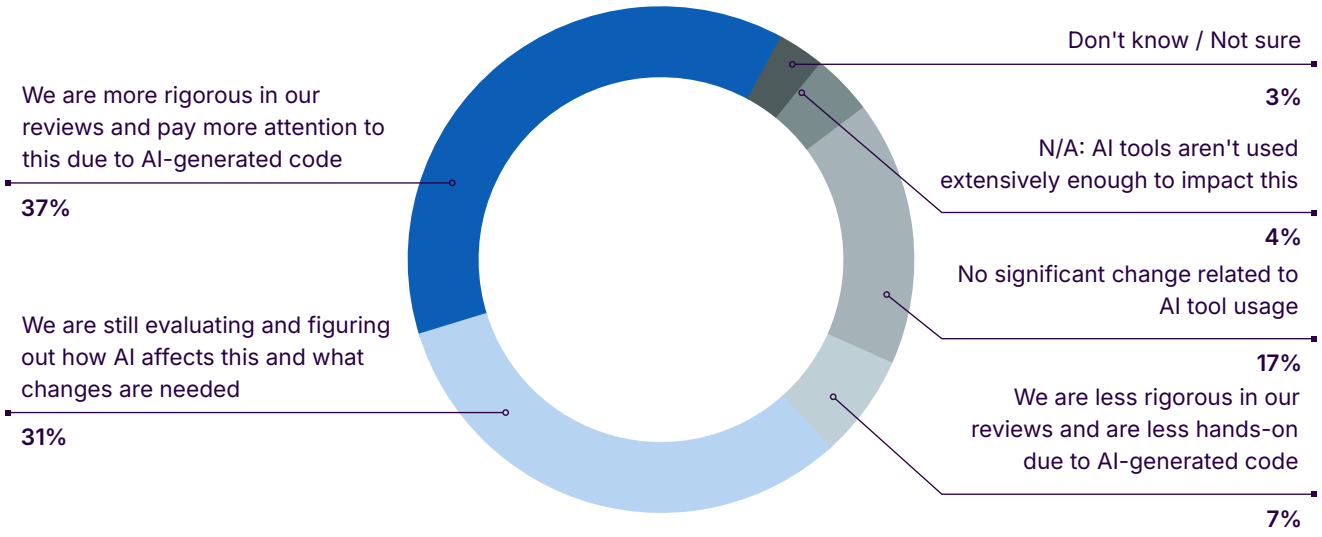
Has there been a change in how you approach the following as a result of the increasing use of AI coding tools? | Choice: Code quality



n=1,149

37% of organizations are more rigorous about code security because of AI

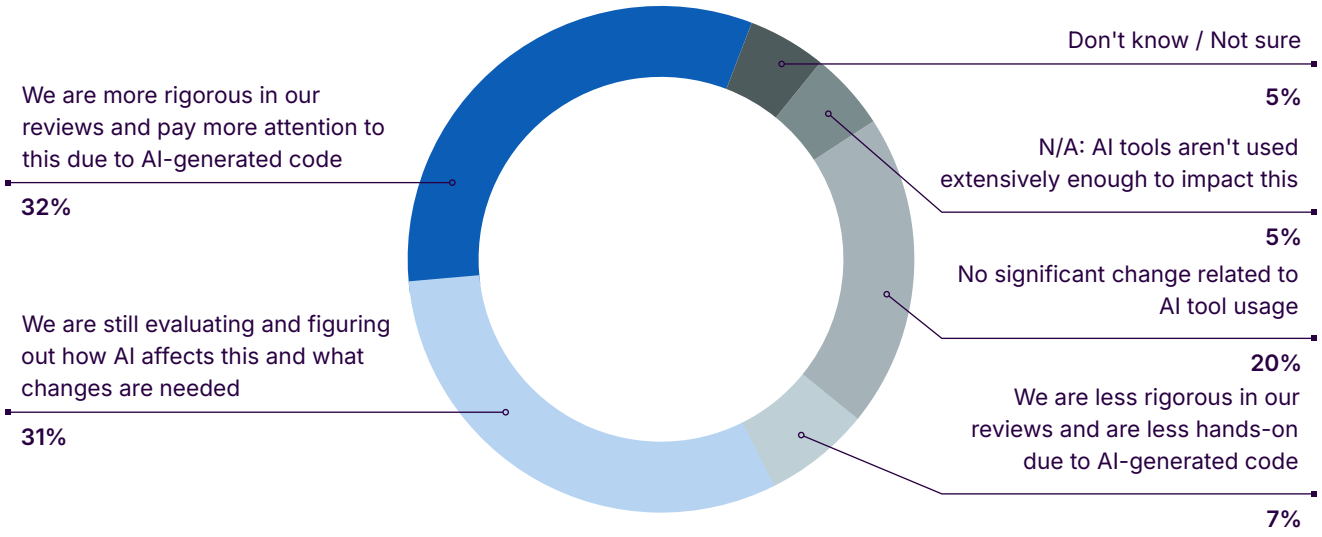
Has there been a change in how you approach the following as a result of the increasing use of AI coding tools? | Choice: Code security



n=1,149

32% of organizations are more rigorous about compliance because of AI

Has there been a change in how you approach the following as a result of the increasing use of AI coding tools? | Choice: Compliance



n=1,149

The takeaway

It's clear that as AI adoption moves from experiment to enterprise staple, organizations are scrambling to build guardrails for security, quality, and compliance. But this new burden of verifying code that "looks correct but isn't reliable" creates a new, hidden drag on teams. This friction—spending time fixing subtle AI errors instead of building features—is essentially a new form of technical debt. Next, we'll explore how AI is fundamentally reshaping what technical debt means.

Trying not to expand technical debt

Most developers have seen both positive and negative effects of AI on technical debt

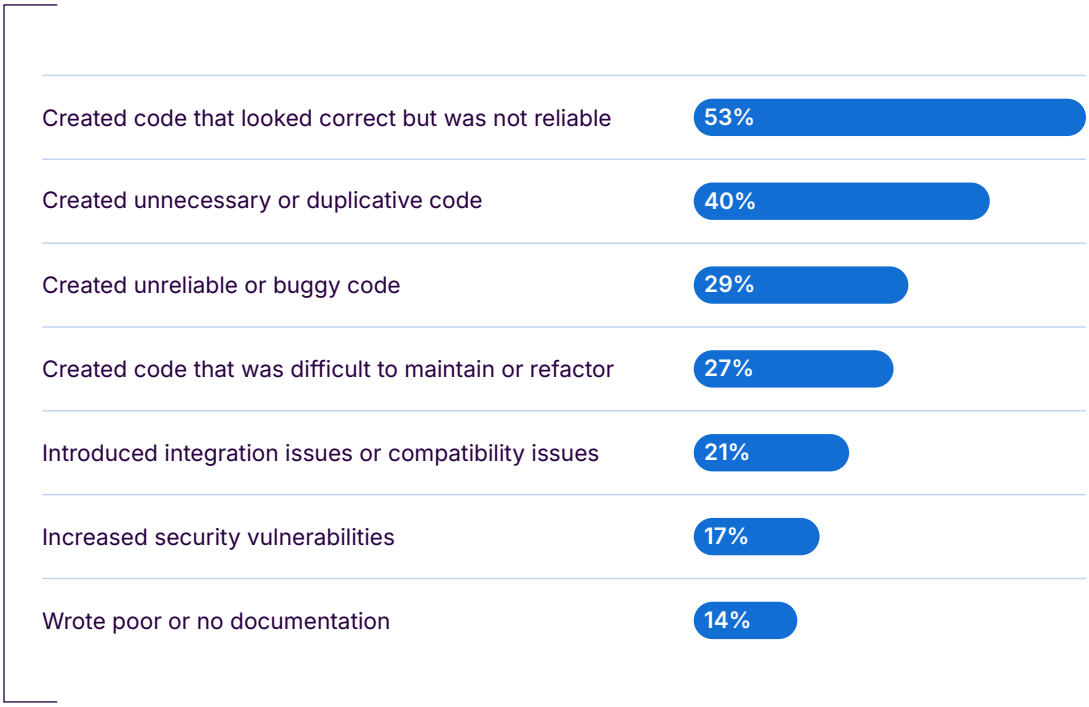
When it comes to technical debt, our study uncovered a complex picture. We found that AI is taking away with one hand and giving back with the other.

Let's start with the detrimental effects. Nearly all developers (88%) report at least one negative impact of AI on their technical debt.

88% of developers report at least one negative impact of AI on technical debt

In which of the following ways, if any, has AI-generated or assisted code impacted your team / company's technical debt?

88%
respondents citing at least one issue



n=1,136

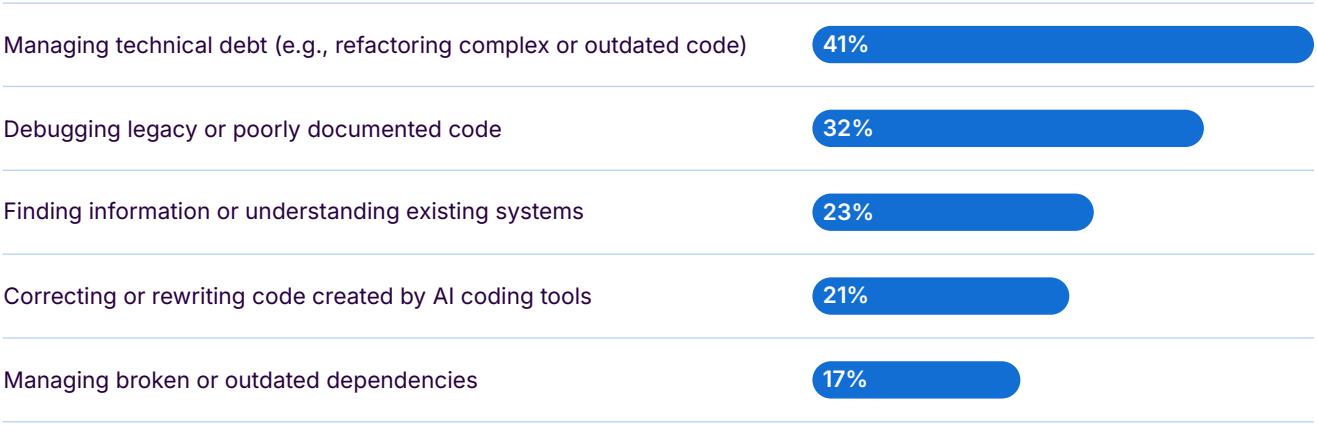
53% of developers attributed a negative impact on their technical debt due to AI creating code that looked correct but was unreliable. This is a particularly pernicious problem because it may create a false sense of security and cause developers to skip thorough review and testing.

The second-biggest negative impact is bloat. 40% of developers say AI has increased technical debt by creating unnecessary or duplicative code. Our LLM personality research confirms that LLMs have inherent tendencies to create verbosity, complexity, and unnecessary technical debt when writing code.

This is a critical issue because "managing technical debt" is already the #1 source of toil for core development tasks, with 41% of developers placing it in their top 5 frustrations. AI, if unmanaged, could pour fuel on an existing fire by generating a high volume of code that's deceptively unreliable.

Top 5 toil work tasks that hinder productivity or increase frustration

Thinking about your current workflows, what are your biggest sources of "toil work" that hinder / sap productivity or increase frustration in your current role?



n=1,149

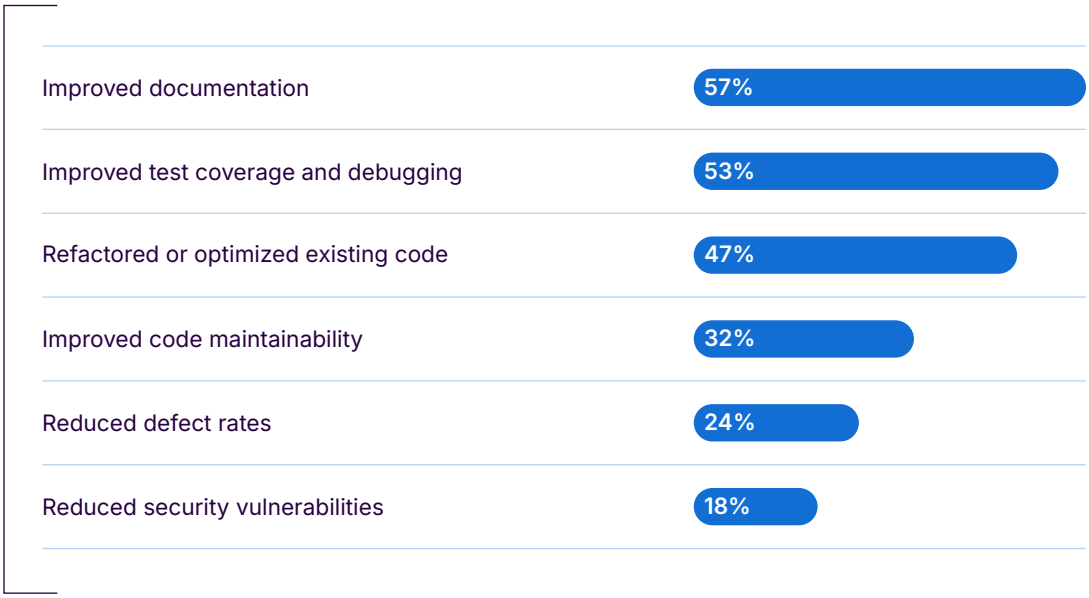
AI is tackling the grunt work

Despite the negative impact on technical debt, nearly all developers (93%) also report at least one positive impact from AI on technical debt as well. Developers are clearly using AI to tackle the most tedious parts of managing technical debt.

93% of developers report at least one positive impact of AI on technical debt

In which of the following ways, if any, has AI-generated or assisted code impacted your team / company's technical debt?

93%
respondents citing at least one benefit



n=1,136

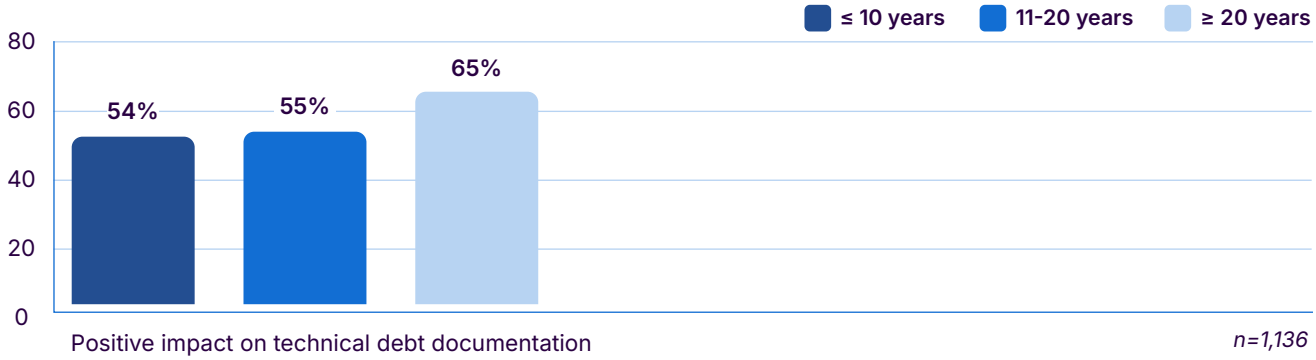
The top positive impacts from AI are all about reducing developer toil work:

- Improved documentation (57%)
- Improved test coverage and debugging (53%)
- Refactoring or optimizing existing code (47%)

This shows developers are intelligently applying AI to the exact problems they hate dealing with. The "improved documentation" stat is especially interesting. Experienced developers, who have likely spent years wrestling with poorly documented legacy systems, seem to value this the most. We found that 65% of developers with over 20 years of experience cited improved documentation as a key technical debt benefit of AI.

Senior developers value impact of AI on technical debt documentation

In which of the following ways, if any, has AI-generated or assisted code impacted your team / company's technical debt?



The data paints a clear picture. AI isn't a simple "fix technical debt" button. It's a powerful, but potentially messy, new collaborator. It's helping developers clean up old messes (like documentation and testing) but creating new, more subtle messes in the process (like unreliable or duplicative code).

The takeaway

It's clear that AI's impact on technical debt is complicated—it's both a powerful clean-up tool and a new source of messy, hidden problems. But this new reality isn't being experienced the same way by everyone on the team. Our data shows a fascinating split in how developers are using AI, what they're using it for, and what they worry about. One of the biggest dividers? Years of experience.

Next, we'll dig into the differing AI use cases and attitudes of junior and senior developers—and what it means for your team's code health.

AI coding and the experience gap

The impacts of AI coding are hitting junior developers harder

Our study reveals a fascinating perception gap: less-experienced developers (≤10 years) are more likely to report productivity benefits from using AI tools than their most-experienced peers (>20 years), but they're also more likely to report that reviewing AI-generated code takes greater effort.

The junior-senior split: How AI is used

We're seeing a clear divergence in how these two groups apply AI.

Less-experienced developers estimate that 45% of their committed code is AI-assisted, slightly more than the 40% estimated by their most-experienced peers. The junior group is also more likely to use newer tools like Cursor and Perplexity, as well as agentic AI, whereas senior developers will tend to rely on more proven AI tools like Copilot.

Less-experienced developers are also more likely than senior developers to use AI for development tasks like explaining existing code, updating functionality, and generating tests. In contrast, their most-experienced colleagues are more likely to use it for code review and assisting development of new code.

AI use case adoption based on years of experience

For which of the following tasks is your team / company using AI coding tools?

	≤ 10 years	11-20 years	≥ 20 years
Assisting development of new code	86%	92%*	90%
Explaining or understanding existing code	80%*	78%	74%
Adding or updating functionality in existing code	78%	78%	68%*
Generating tests	76%	77%	69%*
Writing documentation	74%	75%	73%
Researching technical solutions or exploring APIs/libraries	75%	75%	71%
Refactoring or optimizing existing code	74%	72%	69%
Debugging code (for example, stack trace analysis)	67%	65%	63%
Code review	52%	57%	58%
Translating code from one programming language to another	49%	52%	46%

*n=1,149; *indicates statistical significance*

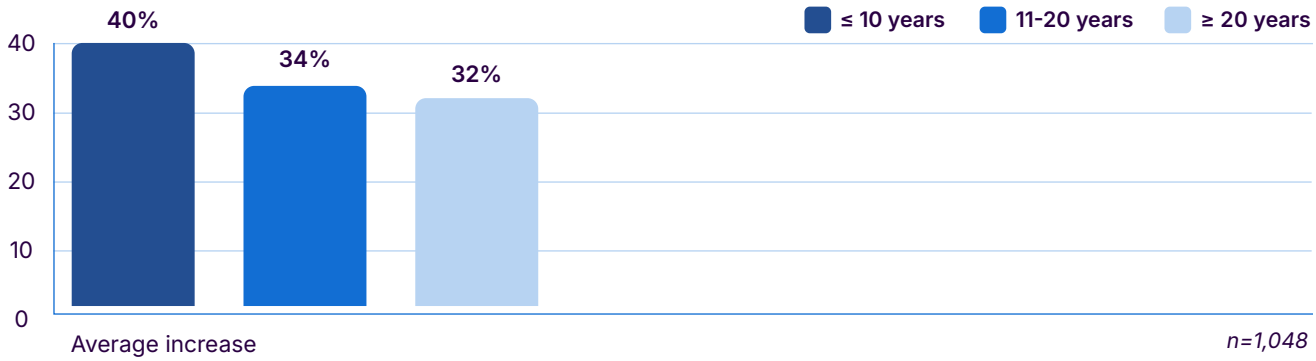
Junior developers report higher productivity

For less-experienced developers, the upside of AI is undeniable. They report significantly higher gains across the board than their senior counterparts:

- Productivity: Junior developers report a 40% average productivity increase versus 32% for their more experienced peers.

Junior developers report higher average increase in personal productivity due to AI

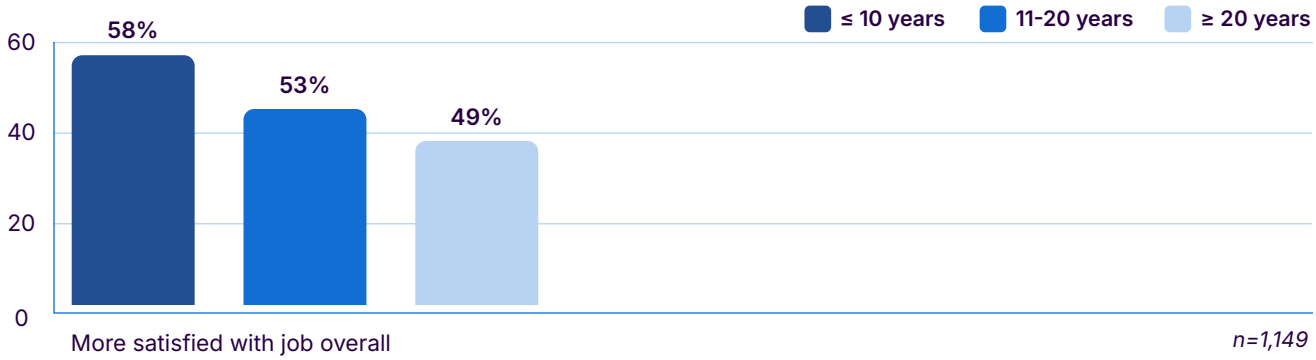
Please estimate the percentage change in your personal productivity due to AI coding tools.



- Job satisfaction: 58% report higher job satisfaction, compared to 49% of senior developers.

Junior developers report higher job satisfaction related to the use of AI/AI-coding tools

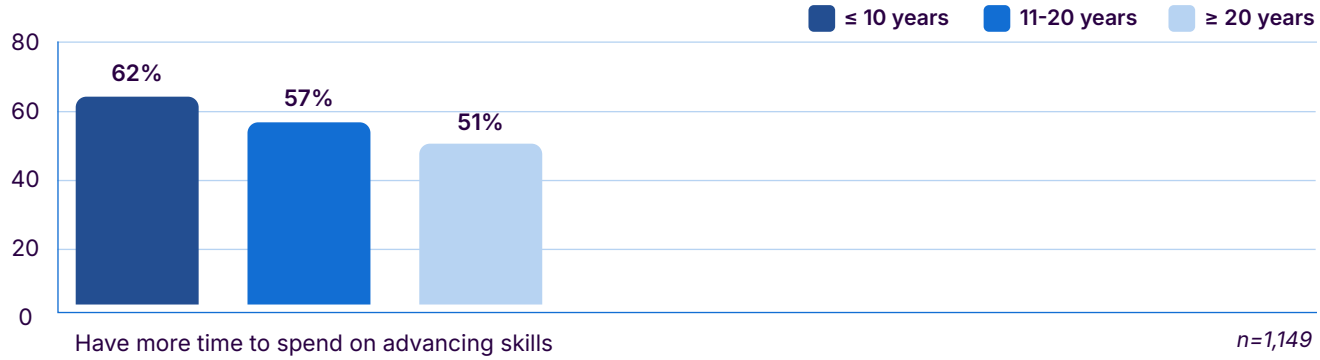
To what extent do you agree with each of the following statements as it relates to the use of AI coding tools and AI-assisted or generated code? | Choice: I'm more satisfied with my job overall since we started using AI



- Skill advancement: 62% say they have more time to advance their skills, far outpacing the 51% of senior developers who agree.

62% of junior developers say they have more time to advance their skills, far outpacing the 51% of senior developers who agree.

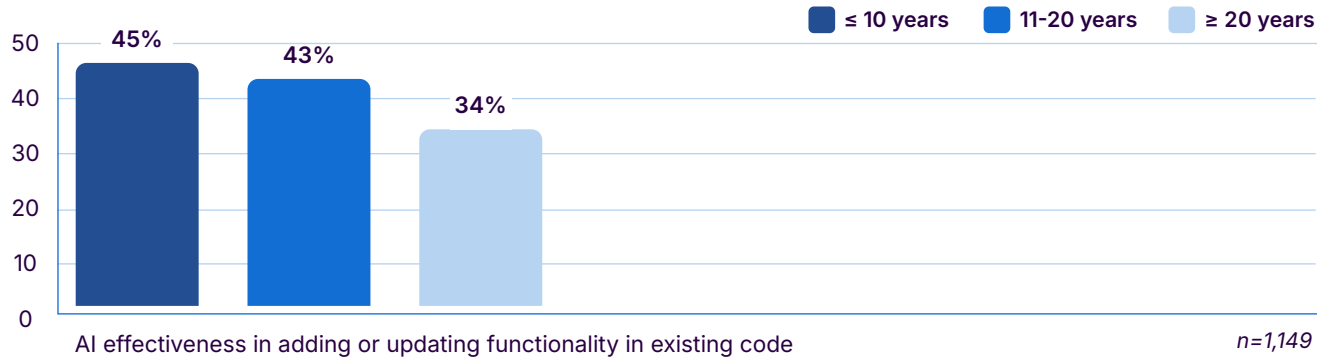
To what extent do you agree with each of the following statements as it relates to the use of AI coding tools and AI-assisted or generated code? | Choice: I have more time to spend on advancing my skills than before I used AI



Less-experienced developers also perceive AI as being more effective in adding or updating functionality in existing code. When asked about AI's effectiveness for this use case, 45% of junior developers said it was effective, compared to only 34% of senior developers.

Junior developers find AI more effective in adding or updating functionality in existing code

How effective are AI coding tools for each of the following tasks you or your team / company has used them for? / Choice: Extremely / Very Effective



Senior developers report fewer concerns

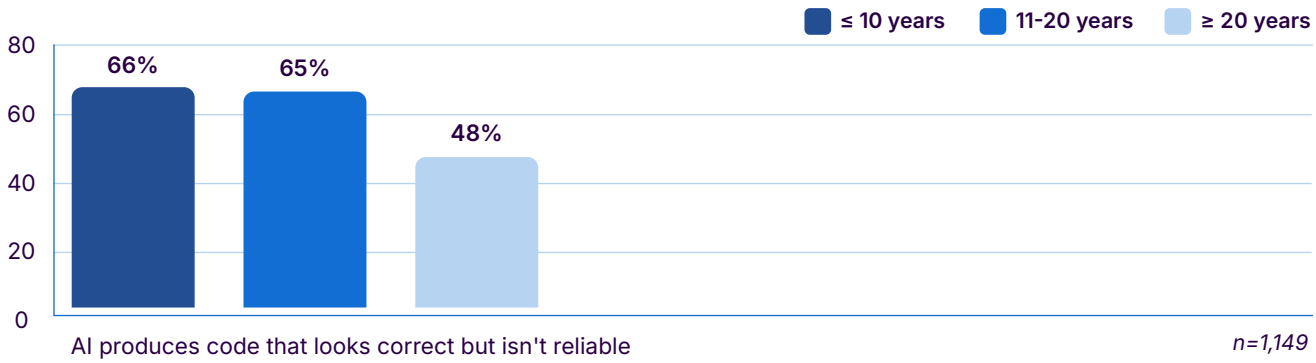
Looking at the findings above, you would be forgiven for thinking that junior developers are generally less concerned about the negative impacts of AI than their senior colleagues. But it turns out that the reverse is true.

Senior developers are significantly less likely to agree with a number of critical concerns:

- Reliability: 48% say AI code "looks correct but isn't reliable" (vs. 66% of junior developers).
- Volume: 32% agree that "too many lines of code are being generated with AI" (vs. 47% of junior developers).

Senior developers are less concerned about AI producing code that looks correct but isn't reliable

To what extent do you agree with each of the following statements as it relates to the use of AI coding tools and AI-assisted or generated code? | Choices: Completely / Somewhat agree

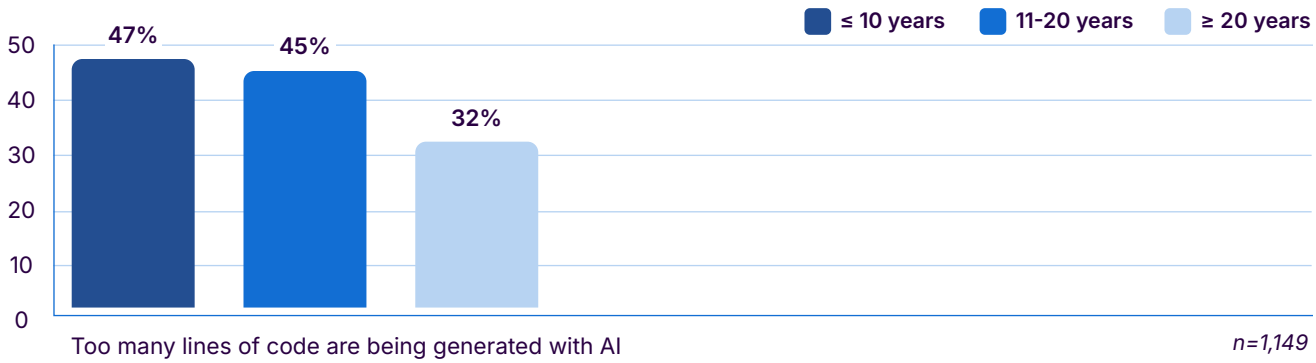


AI produces code that looks correct but isn't reliable

n=1,149

Senior developers are less concerned about AI generating too many lines of code

To what extent do you agree with each of the following statements as it relates to the use of AI coding tools and AI-assisted or generated code? | Choices: Completely / Somewhat agree



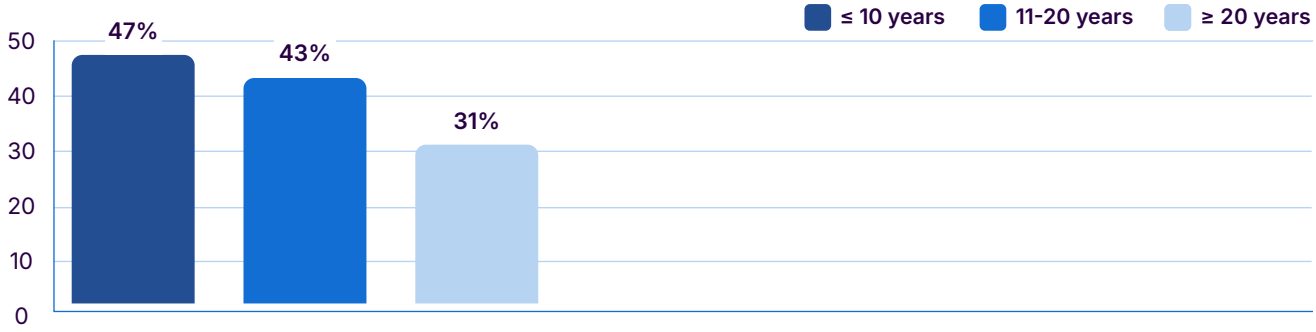
Too many lines of code are being generated with AI

n=1,149

- Context: 31% agree that AI "makes complex coding tasks more difficult because it lacks context" (vs. 47% of junior developers).

Senior developers are less concerned about AI making complex coding tasks more difficult due to lack of context or understanding

To what extent do you agree with each of the following statements as it relates to the use of AI coding tools and AI-assisted or generated code? | Choices: Completely / Somewhat agree

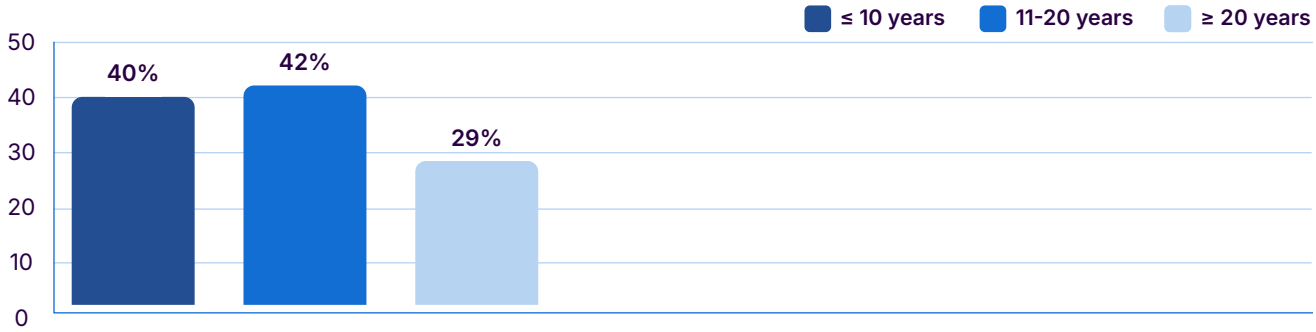


AI makes completing complex coding tasks more difficult because it often lacks context or understanding *n=1,149*

This directly translates into review effort. Developers with 10 years or less of experience are significantly more likely (40%) to say that reviewing AI-generated code requires more effort than reviewing code written by a developer. Only 29% of developers with ≥ 20 years of experience feel the same way.

Senior developers are less likely to say that reviewing AI-generated code requires more effort compared to developer-written code

Does reviewing AI generated code require more or less effort than reviewing code written by developers?

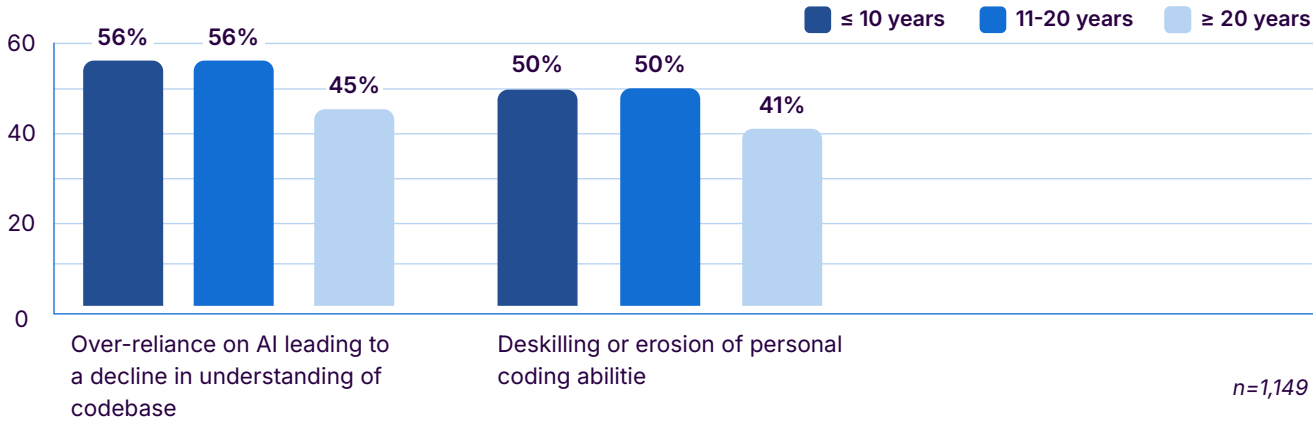


Reviewing AI-generated code requires more effort than developer-written code *n=1,138*

Senior developers also express less anxiety about the long-term impact of AI on their skills, with 41% concerned about the erosion of their personal coding abilities (vs. 50% of junior developers) and 45% worried about a decline in codebase understanding (vs. 56% of junior developers).

Senior developers are less worried about the impact AI is making on their skills & abilities

How concerned are you about each of the following when it comes to AI-assisted or generated code today? | Choice: Extremely concerned / Very concerned



n=1,149

The impacts of experience

The data tells a clear story. Less-experienced developers are leaning on AI to understand and write code, getting a massive productivity boost, but also the first to feel the pain when that code is buggy, verbose, or hard to maintain. More-experienced developers, meanwhile, appear to be more skeptical of its output, applying it to tasks like prototyping, where speed is key and the code isn't expected to be perfect. They seem less concerned about the potential negative impacts of AI code, perhaps because they are more targeted in how they use AI tools, or because they believe in their own abilities to catch and correct issues with AI code before they cause problems.

The takeaway

It's clear that a developer's years of experience fundamentally shape their relationship with AI, from the tools they choose to the friction they feel. But experience isn't the only major divide our study uncovered. The size of an organization also plays a critical role in shaping strategy and outcomes. Next, we'll explore why smaller companies are reporting bigger productivity gains from AI, while large enterprises are moving more cautiously, focusing on rigorous compliance.

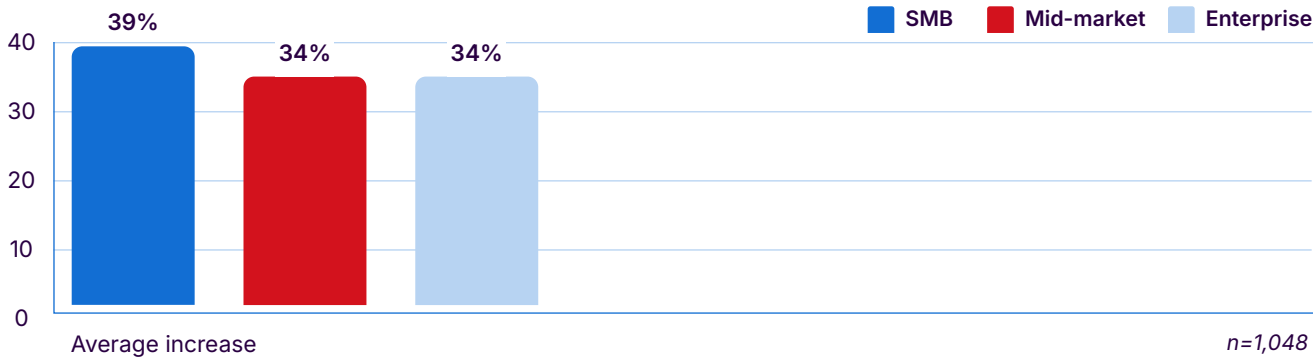
How enterprises and small businesses are approaching AI

Small businesses move faster with AI, but enterprises get better code

According to our study, developers at small-to-medium businesses (SMBs under 200 employees) report a 39% increase in personal productivity—slightly outpacing other developers who report a 34% lift.

SMB developers report a higher average increase in personal productivity due to AI

Please estimate the percentage change in your personal productivity due to AI coding tools.



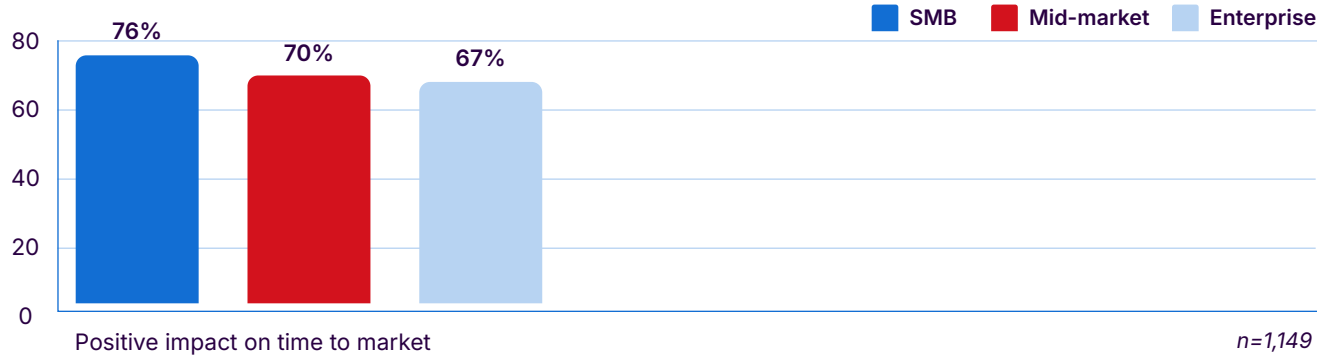
While nearly all developers are seeing productivity gains, the data reveals two distinct realities. Smaller teams are sprinting ahead with accessible tools to boost raw speed, while enterprises are taking a more measured approach, focusing on governance and long-term code health.

The trade-off between speed and quality

It's clear that smaller organizations are capitalizing on the immediate velocity AI offers. Beyond the higher average productivity gains, 43% of SMB developers report being "much more productive" personally, compared to only 36% of enterprise developers. This focus on speed translates directly to business outcomes, with smaller organizations more likely to report gains in time-to-market (76%, compared to 67% of enterprise developers).

Smaller organizations are more likely to report gains in time-to-market

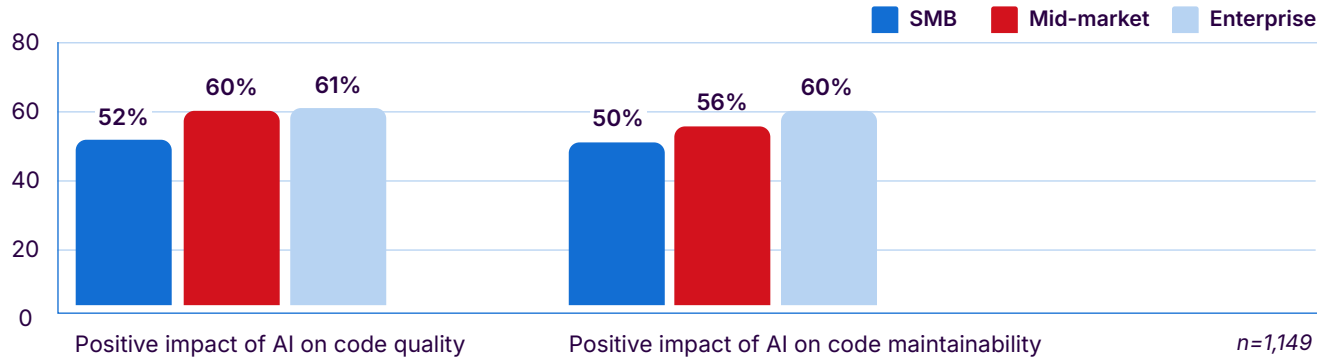
What impact has AI-generated or assisted code had on your team / company for each of the following? | Choices: Very positive / Somewhat positive impact



However, enterprises are seeing a different upside. While they report lower raw speed gains, they are more likely to report improvements in code quality and maintainability (almost 20% more often than SMB). This suggests that larger organizations are leveraging AI to improve the quality of their massive existing codebases.

Enterprises report higher positive impacts on code quality and code maintainability

What impact has AI-generated or assisted code had on your team / company for each of the following? | Choices: Very positive / Somewhat positive

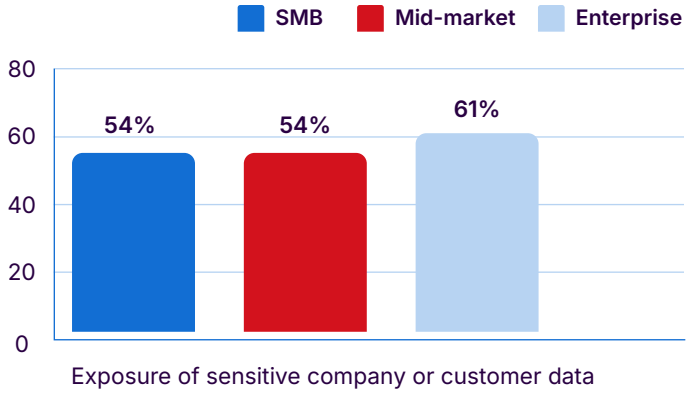


The benefits of enterprise governance

Enterprises appear to be managing AI risks through rigorous governance. They are statistically more concerned about the exposure of sensitive data (61%).

Enterprises are more concerned about the exposure of sensitive data when it comes to AI-generated or assisted code

How concerned are you about each of the following when it comes to AI-assisted or generated code today? | Choice: Extremely concerned / Very concerned



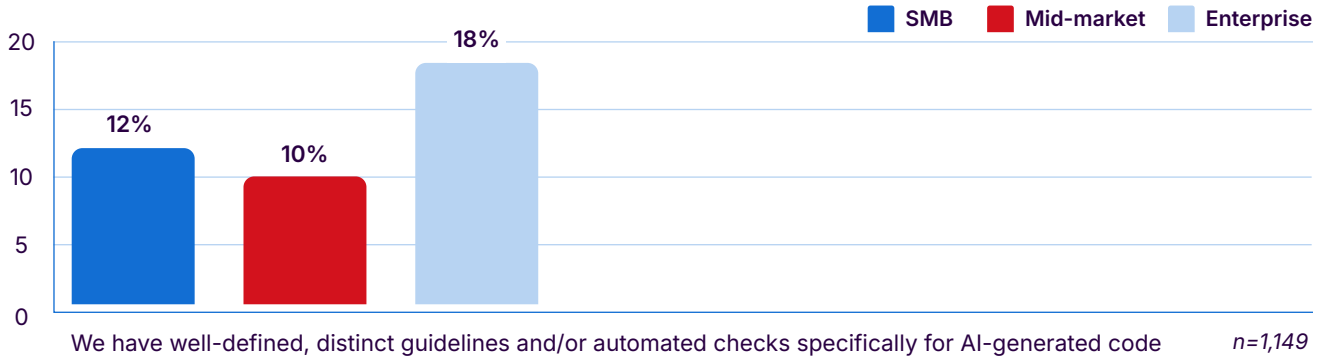
n=1,149

This is driving them to implement stricter controls:

- Formal policies: Enterprises are more likely to have well-defined, distinct guidelines or automated checks specifically for AI-generated code (18%) compared to SMBs (12%).

Enterprises are more likely to have distinct guidelines and / or automated checks specifically for AI-generated code

Does your [org] have specific guidelines, policies, or quality gates for the review, testing, and acceptance of AI-generated code that differ from those for [dev-written] code?

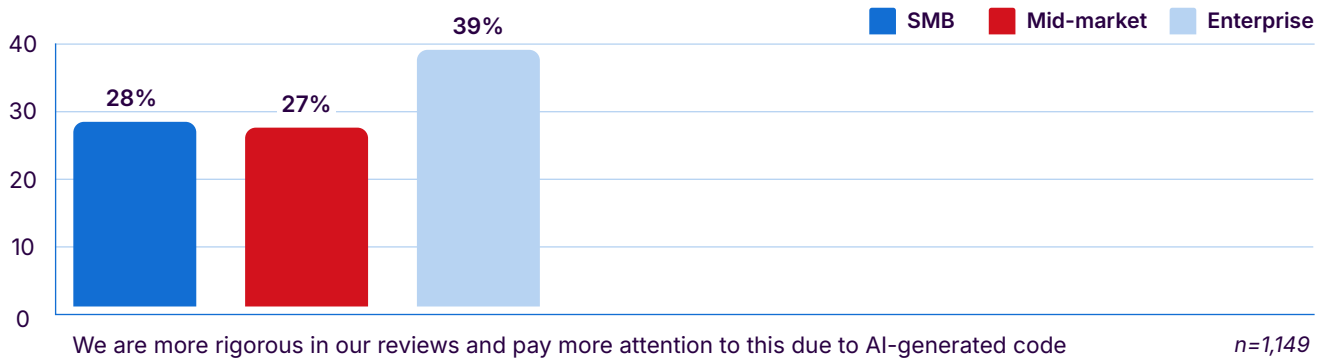


n=1,149

- Compliance rigor: When it comes to compliance reviews for AI-generated code, 39% of enterprise respondents say they are more rigorous now, compared to only 28% of SMBs.

Enterprises are the most rigorous and pay more attention to AI-generated code when it comes to compliance

Has there been a change in how you approach the following as a result of the increasing use of AI coding tools?



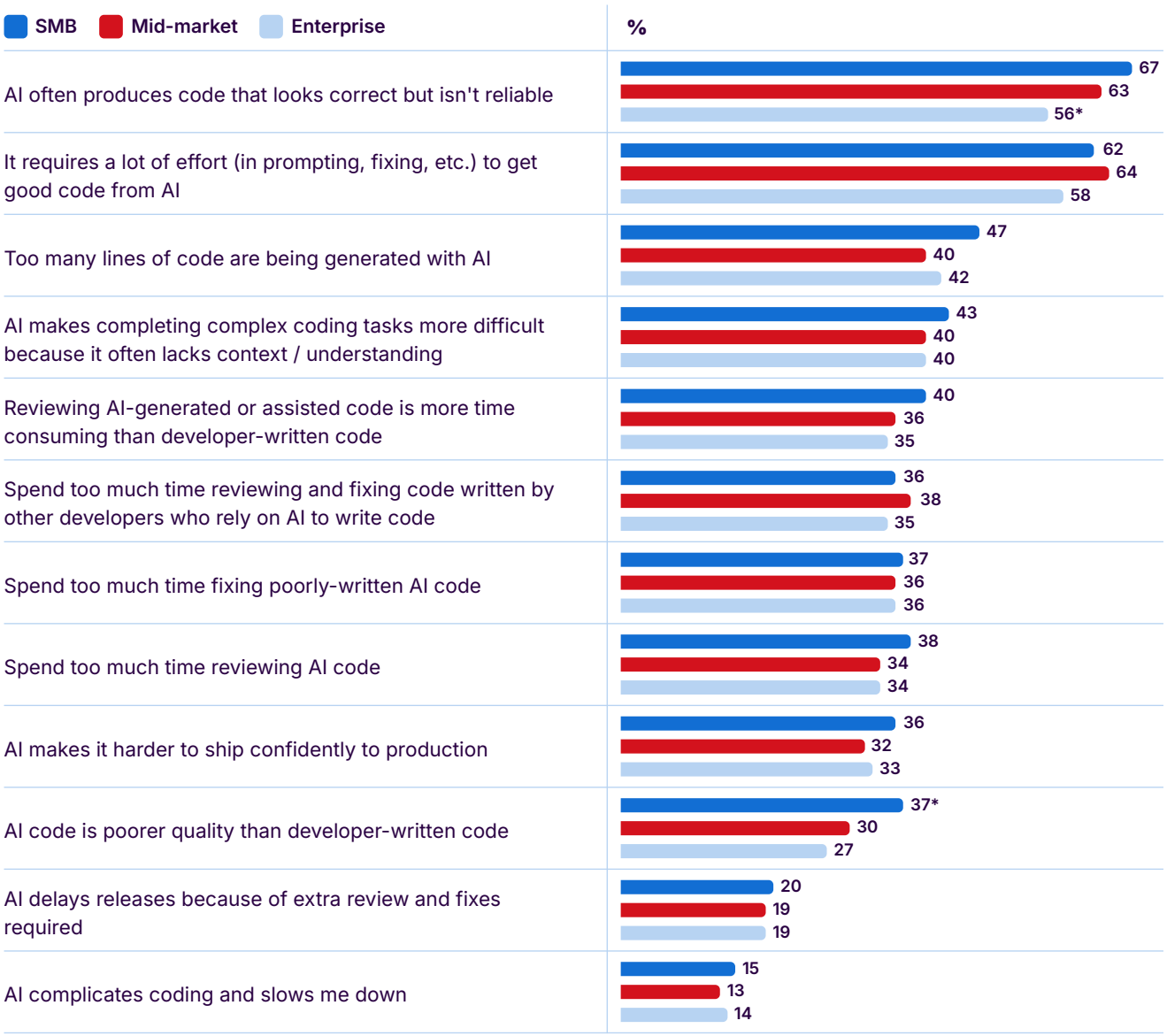
We are more rigorous in our reviews and pay more attention to this due to AI-generated code *n=1,149*

Small and medium businesses report more negative impacts

Another strong indicator that enterprises are getting better ROI on AI? SMBs often report up to 11 percentage points higher negative impacts due to AI. Perhaps because they may lack the formal guardrails of larger orgs, they experience more acute friction managing AI code.

SMBs consistently report higher negative impacts due to AI

To what extent do you agree with each of the following statements as it relates to the use of AI coding tools and AI-assisted or generated code? | Choices: Completely agree / Somewhat agree



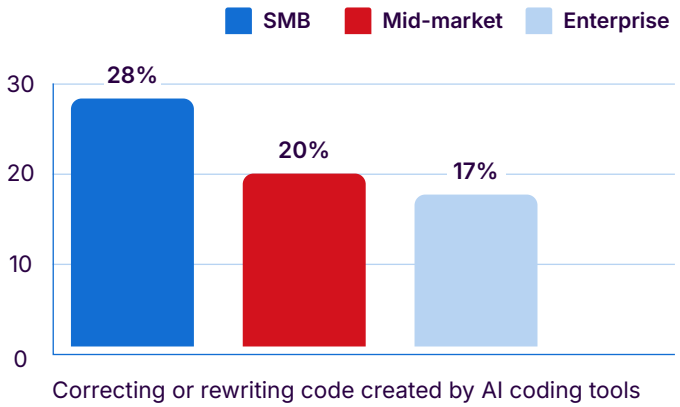
*n=1,149; *indicates statistical significance*

The most frequently reported negative impact of AI on code is that it looks correct but isn't reliable. Notably, 67% of SMB developers agree with that statement, compared to 56% of enterprise developers.

Perhaps most telling, SMB developers are significantly more likely to cite "correcting or rewriting code created by AI coding tools" as a top source of toil work (28%) compared to enterprise developers (17%).

SMB developers are more likely to cite "correcting or rewriting the code created by AI coding tools" as one of the top sources of toil work

Thinking about your current workflows, what are your biggest sources of "toil work" that hinder / sap productivity or increase frustration in your current role?



n=1,149

This indicates that while SMBs are moving fast, they are paying for it in technical debt and rework. They are also less likely to use automated code review tools to check AI-generated code, which may exacerbate the issue.

The data paints a picture of two different trajectories. SMBs are volume-driven adopters using low-friction tools to boost personal speed, but they face increased developer toil correcting unreliable output. Enterprises are deliberate, risk-averse adopters prioritizing data security and systemic improvements.

The takeaway

While AI adoption is high across the board, the experience of that adoption is clearly different. Enterprises are investing in the process and governance to manage AI-generated code, which leads to higher-quality, more maintainable code. SMBs are reaping the immediate speed benefits but are feeling the pain in verification and rework. This highlights a critical lesson for everyone: generating code faster is only half the battle. The real value comes from being able to trust and verify that code efficiently.

SonarQube: The essential verification layer for AI code

SonarQube users get more ROI out of AI coding tools than other developers

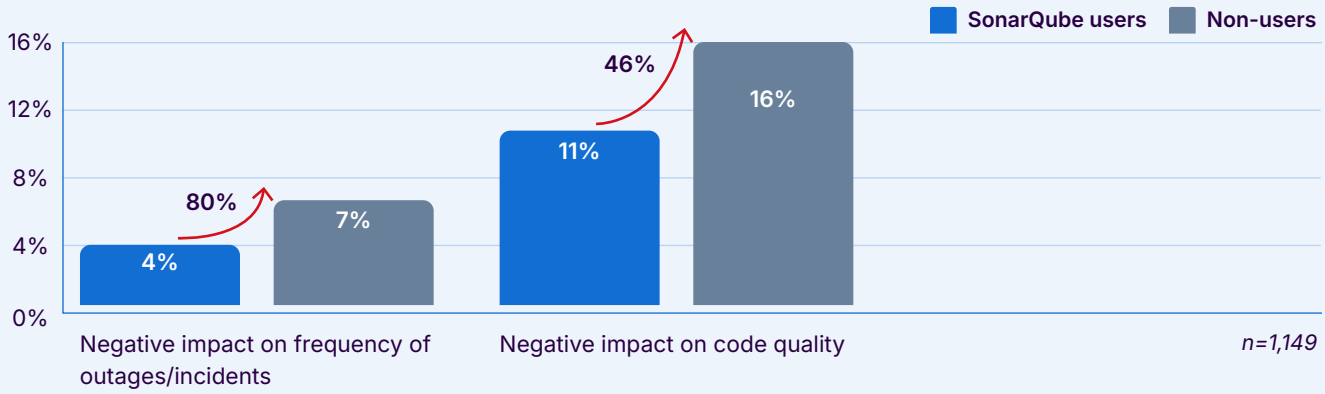
If there's one thing this research has shown, it's that AI does not fix a broken system. It simply amplifies what is already there.

If you have a high-trust, well-architected system with a robust testing and verification platform, AI will amplify your efficiency. But if you have a low-trust, chaotic system, AI will only amplify that chaos. It will give you more low-quality, untrusted, "looks correct but isn't" code, faster than ever before.

In light of this, it's no surprise that developers not using SonarQube are 80% more likely to report that AI adoption led to a higher frequency of outages and incidents, and 46% more likely to say AI had a negative impact on their overall code quality.

Non-users of SonarQube are more likely to report negative impact of AI on outage frequency and code quality

What impact has AI-generated or assisted code had on your team / company for each of the following? | Choice: Very / somewhat negative impact



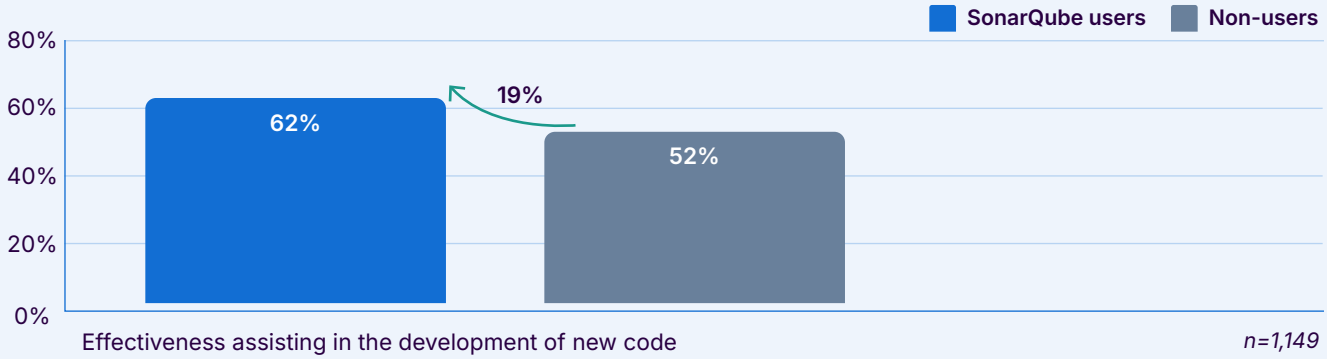
This data points directly to the solution to the "AI accountability crisis." When teams adopt AI to "vibe" (generate code) alongside a strong, independent way to "verify" (check that code), they aren't just getting productivity gains—they're actively decreasing their risk.

SonarQube users find AI more effective for core tasks

The data shows that SonarQube users aren't just avoiding the pitfalls; they're actively getting more value from their AI tools. We saw a clear pattern: SonarQube users consistently rated AI's effectiveness 19% higher for assisting in the creation of new code.

SonarQube users consistently report higher levels of AI effectiveness assisting in the development of new code

How effective are AI coding tools for each of the following tasks you or your team / company has used them for? | Choice: Extremely / Very Effective



When developers have a trusted partner like SonarQube providing real-time, actionable code intelligence, they can use AI tools with more confidence. They're not just "vibe coding" and hoping for the best; they're verifying as they go, which makes the whole process more effective.

Better effectiveness leads to better strategic outcomes

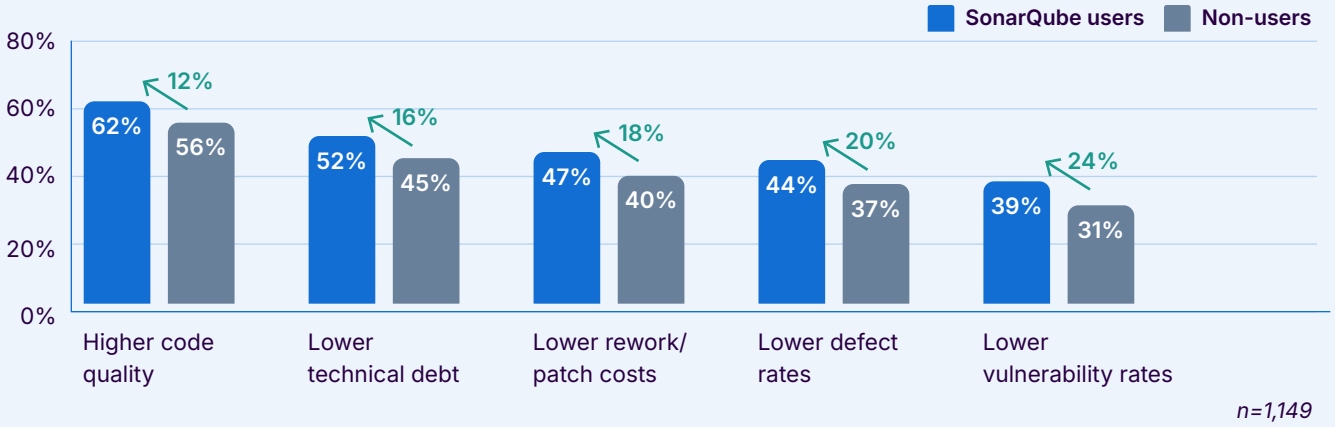
This increased effectiveness isn't just a nice to have. It translates directly into the strategic outcomes that engineering leaders and CIOs care about: higher quality, lower risk, and reduced costs.

When we asked about the positive impacts of AI adoption, Sonar users were:

- 24% more likely to report a positive impact on vulnerability rates.
- 20% more likely to see a positive impact on defect rates.
- 18% more likely to see a reduction in rework and patch costs.
- 16% more likely to report a positive impact on technical debt.
- 12% more likely to report a positive impact on overall code quality.

Sonar users report higher positive impact of AI on the following activities

What impact has AI-generated or assisted code had on your team / company for each of the following?



n=1,149

This data provides a clear pathway to help developers adopt AI coding tools safely and build code everyone can trust. Simply generating code faster doesn't lead to real productivity if it creates a massive verification bottleneck or, worse, introduces new risks that show up as production outages. Real productivity comes from speeding up the entire development lifecycle.

SonarQube users are seeing better results because they have the "verify" part of the equation built in. By integrating automated code review for both developer-written and AI-generated code directly into their workflow, they catch issues earlier, reduce the burden of manual review, and ensure that all code meets their quality and security standards before it ever gets to production.

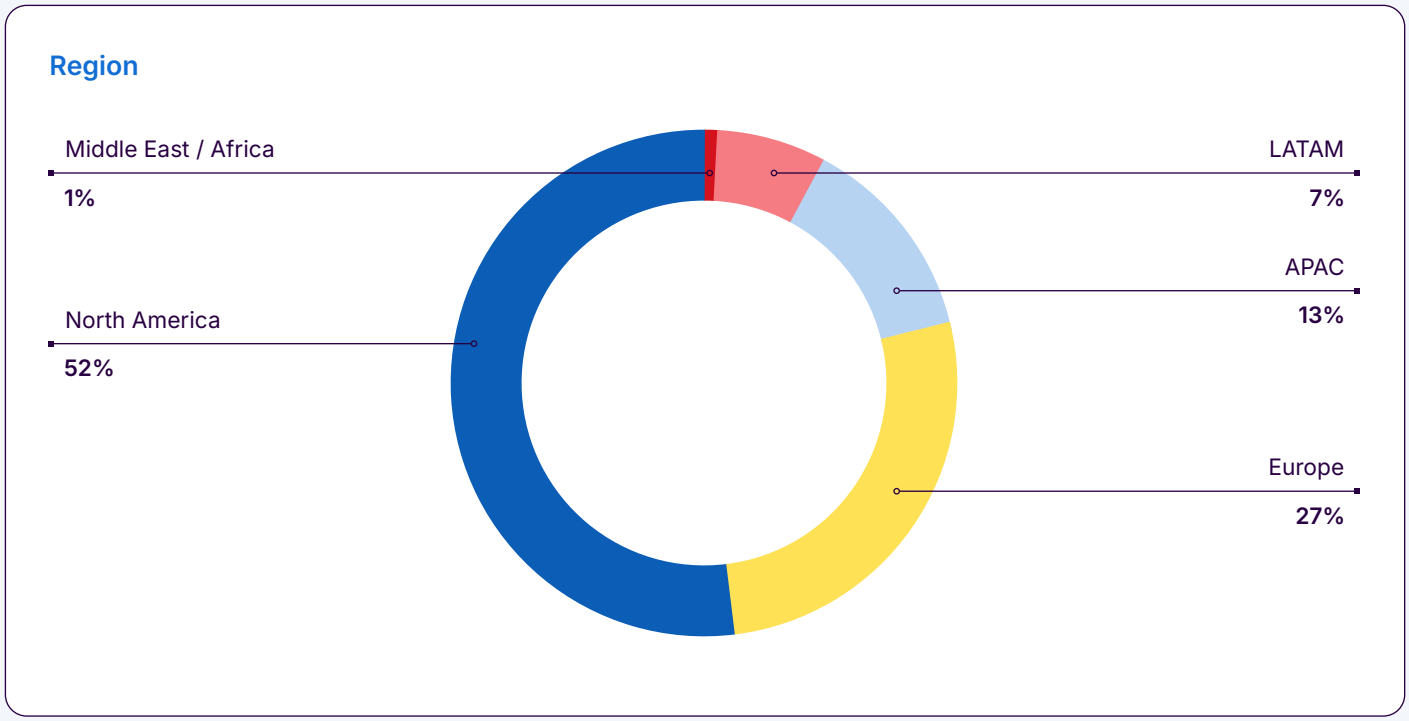
Ultimately, AI is a powerful collaborator, but it's not infallible. The organizations winning with AI are the ones who treat it that way—empowering their teams to vibe, then verify.

Appendix: About our survey demographics

To understand the current state of software development, we surveyed 1,149 technology professionals from around the globe. We wanted to ensure our data reflected the reality of modern engineering teams, so we gathered insights from a diverse mix of developers, managers, and executives working across a wide range of industries and company sizes.

Here's a closer look at who we surveyed.

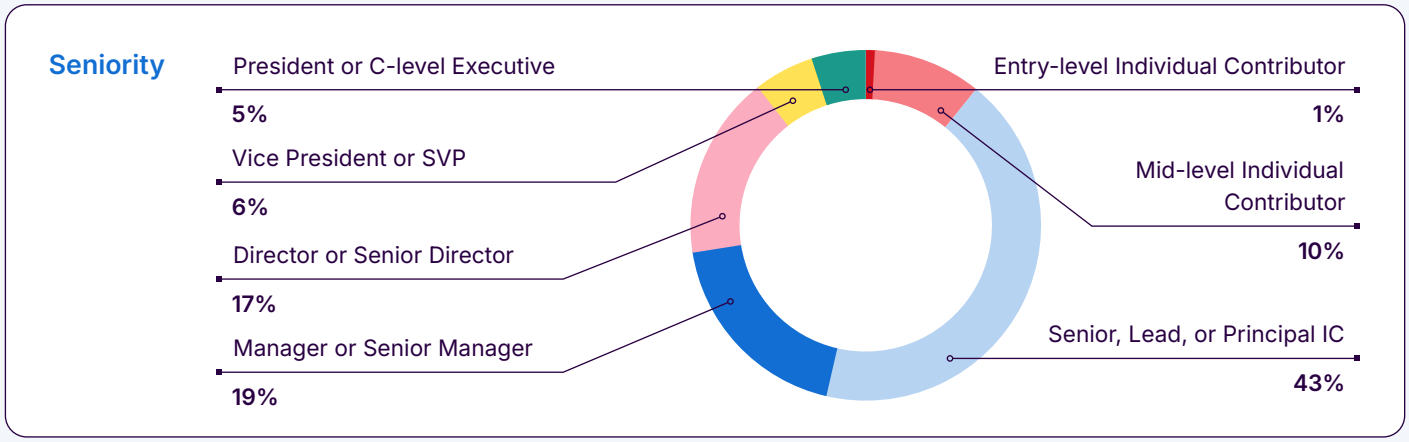
A global perspective



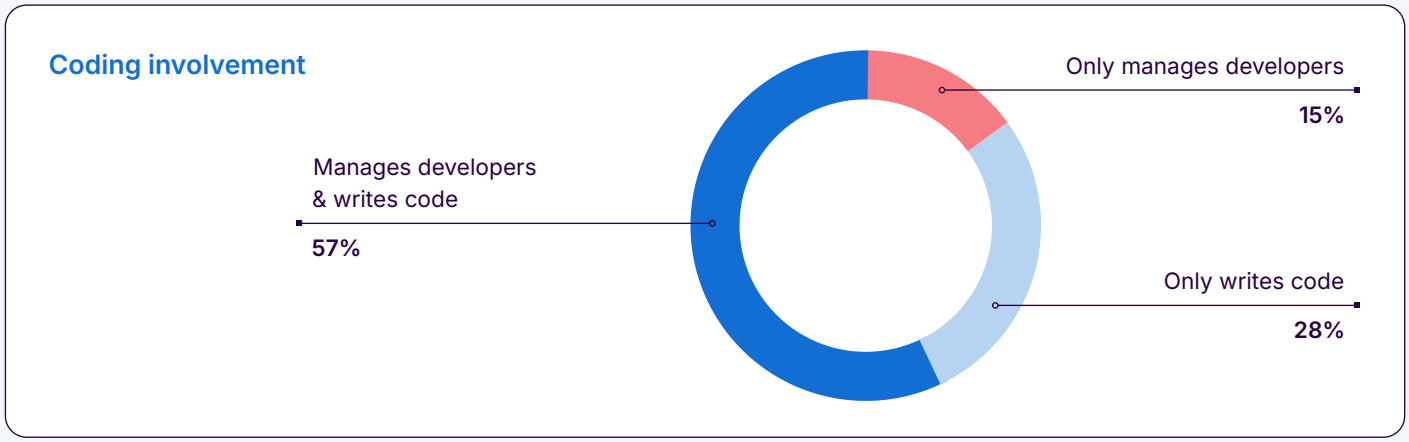
Our respondents come from all over the world, giving us a truly global view of the developer landscape. While just over half (52%) are based in North America, we have significant representation from Europe (27%) and the APAC region (13%), with the remainder from Latin America, the Middle East, and Africa.

Experienced and hands-on

We tapped into professionals across a wide band of experience and seniority.



We also captured the perspective of engineering leadership. Nearly half of our sample holds a management title, ranging from Managers and Directors to VPs and C-level executives.

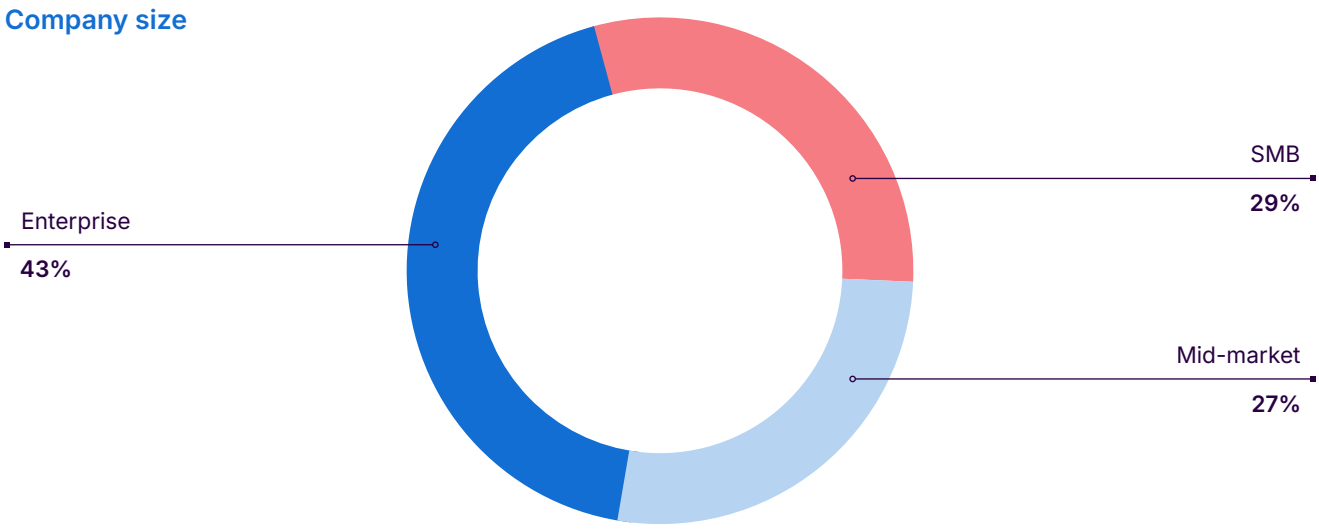


When it comes to their daily work, the vast majority are still deep in the code.

- 57% of respondents both write code and manage developers.
- 28% are focused solely on writing code.
- 15% focus exclusively on managing developers.

This mix ensures our findings reflect both the tactical realities of writing code and the strategic challenges of managing it.

Organizations of all sizes



We wanted to understand how development happens at all scales, and our demographics reflect that. Nearly half of our respondents (43%) work at enterprises with over 1,000 employees. Another 27% work at mid-market companies (201–1,000 employees), while the remaining 29% work at SMB organizations (1–200 employees).

Diverse industries and applications

While the majority of our respondents (62%) work in the technology sector (software, hardware, and services), we also heard from professionals in:

- Financial Services & Insurance (9%)
- Healthcare & Life Sciences (5%)
- Retail & E-commerce (5%)
- Manufacturing, Government, and other key sectors.

The type of software they are building is just as varied. About one-third (28%) are building customer-facing web or mobile apps, while others are focused on commercial software (29%), custom client applications (19%), and internal tools (18%).

Analyst's note on data interpretation

Percentages in this report may not sum to exactly 100% due to rounding or multi-select questions. All data is based on the total sample of 1,149 respondents unless otherwise noted.