The AI adoption paradox

A data-first brief on adoption vs. trust, speed, safety, and value

Written by COSINE

Executive summary

In 2025, use of AI in software development is marked by two paradoxes: the paradox of mistrust, and the paradox of inefficiency.

The paradox of mistrust: this year the share of developers using AI tools has risen to 84%, yet at the same time confidence in the accuracy of AI has fallen to 33%.

The paradox of inefficiency: despite growing AI adoption, backlogs are still high, with technical debt remaining by far the top frustration for leaders at 62%. Meanwhile security debt persists in 42% of applications and 71% of organisations, with 48% of known vulnerabilities still open after 1 year.

Two trends are driving these dynamics.

First, off-the-shelf AI code has been proven to lead to less secure outcomes. Peer-reviewed studies show that users of basic AI assistants not only produce less secure code, but feel more confident about its security, versus those who don't use AI. Separately, a large study in 2024 evaluated sixteen code-generating models and reported package hallucination rates of 22% for open-source models (versus 5% for commercial models) with 205,000 unique fake package names observed.

Second, only partial deployment of AI creates bottlenecks and erodes trust. Developers use AI heavily for writing code (82%), search (68%), and debugging (57%), but far fewer use it for documentation (40%), testing (27%) or commit and review (13%). Reviewers then face proposals without tests, policy results, or clear rationale, and the leading blockers become trust (66%) and codebase context (63%).

There are reasons not to lose hope.

AI still has clear benefits to software developers: one enterprise-scale rollout found 9% more pull requests per developer, a 15% higher merge rate, and an 84% increase in successful builds, while a separate security study reduced median time to remediation by about 67%.

The secret to success is letting custom-built AI deliver orchestration rather than just autocomplete.

Policy-aware agents should open evidence-rich pull requests, run CI and security checks, and attach audit trails so reviewers can approve with confidence and throughput improves. This approach both addresses mistrust by providing context and improves throughput by enabling faster human review.

Tech leaders can drive tangible progress in 90 days.

Key steps are: 1) baseline pull request, CI and security metrics; 2) pilot two high-volume flows such as tests and small fixes with policy-aware agents inside your boundary; 3) require proof on every AI-touched change; and 4) scale once merge rate rises, time to merge falls, build success rises, and aged issues decline.

About Cosine

Cosine provides autonomous, policy aware engineering agents that work through the software delivery pipeline. The agents open evidence rich pull requests, generate and run tests, satisfy security and compliance checks, and attach clear rationale so reviewers can approve with confidence. Cosine deploys inside your boundary and integrates with your existing repositories, continuous integration, and security tooling.

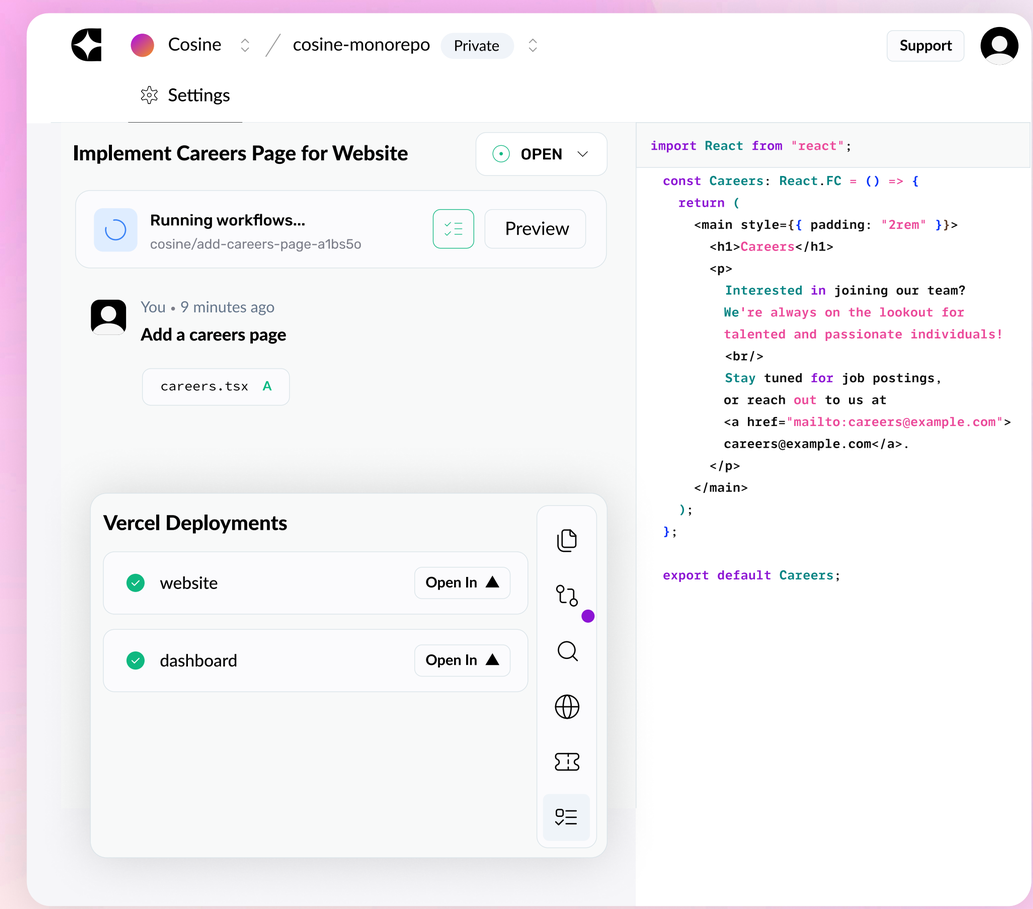
Cosine serves engineering leaders who need measurable throughput in complex or regulated settings. Typical owners include Heads of Engineering, Platform Engineering, and Application Security. Teams use Cosine for high volume flows such as test generation and maintenance, small bug fixes, dependency and lockfile updates, flaky test repair, documentation at scale, and targeted security remediation. The system is asynchronous and queue driven, so it clears backlogs without interrupting developer focus.

The benefit is felt in outcomes rather than anecdotes. Organisations adopt Cosine to increase pull request throughput and merge rate, raise build success, shorten time to remediation, shrink aged technical and security debt, and maintain a complete audit trail with data kept inside their boundary. Because Cosine orchestrates the tools teams already use, time to value is short and change management is straightforward.

Cosine is available for free as a cloud service online, while enterprises can also explore proof of concept trials including virtual private cloud, on premises, and fully air gapped deployments. This lets teams validate value in their own environment and move to scale with confidence.

Give it a try

Book a demo

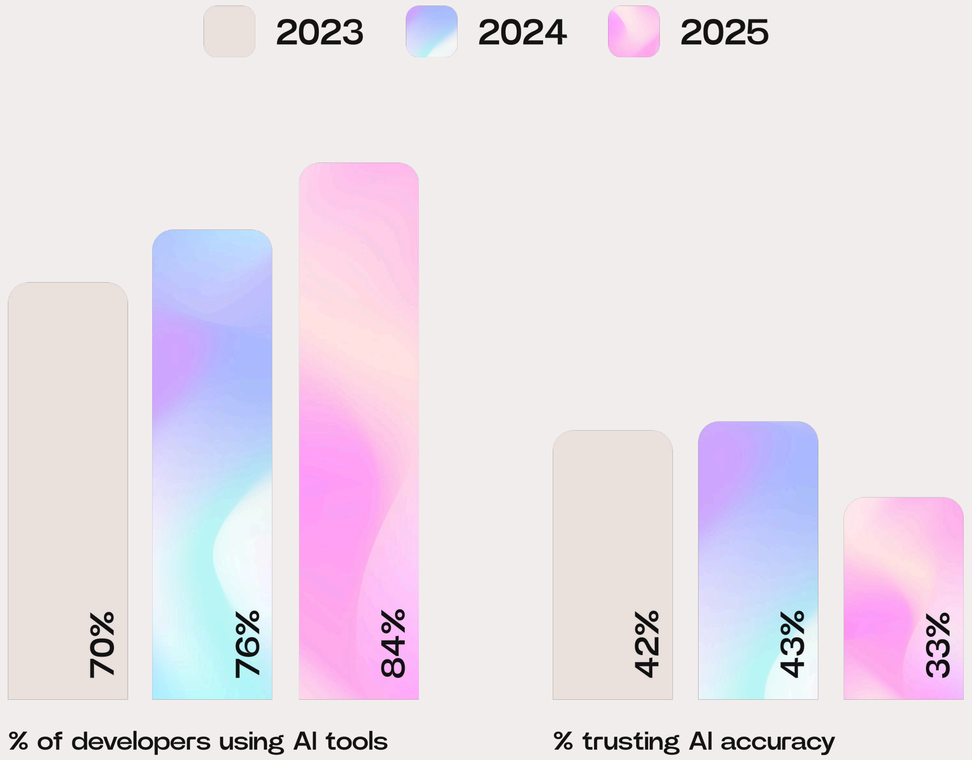


The paradox of mistrust

AI use is growing, but trust in AI is declining

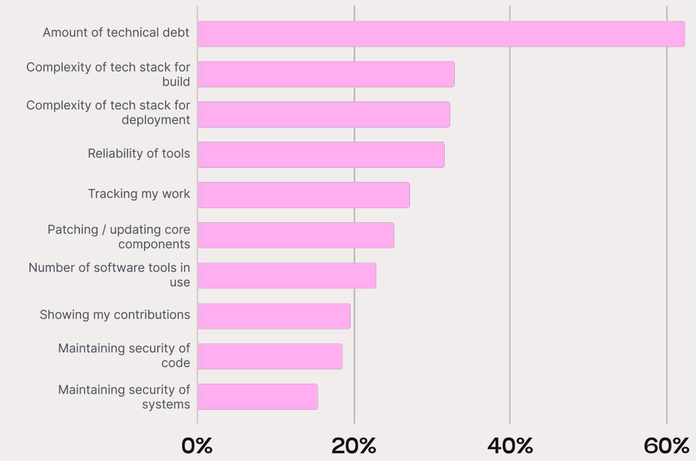
Use of AI in software development is marked by two seemingly counteracting trends. Today, 84% of developers are using AI tools - up from 70% two years ago. But at the same time, confidence in the accuracy of these tools is falling, and now stands at just 33%. This pattern captures the adoption paradox in a single statistic set. **Usage increases while trust erodes.**

The point of hesitation is sitting at the delivery gates. Teams are comfortable drawing on AI inside the code editor for drafting, searching, and minor fixes. Hesitation emerges when proposed changes must pass tests, peer review, security policy, and formal change management. As AI output moves from boilerplate toward business-critical code, reviewers confront missing repository context, limited provenance, and incomplete audit trails. These gaps lengthen review cycles and reduce approval rates.



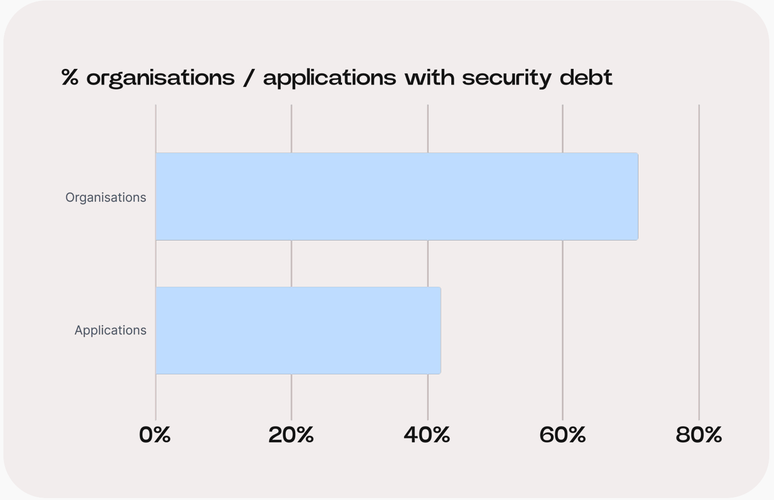
Source: 2024 Developer Survey, Stack Overflow

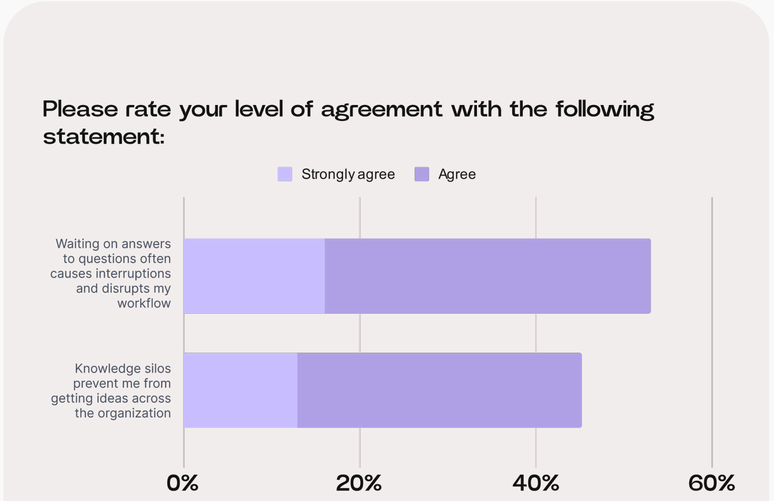
The paradox of inefficiency | AI use is rising, but so are technical and security backlogs



There is a second paradox at play: the paradox of inefficiency. AI improves drafting speed, but the gains have not yet translated into higher throughput or smaller backlogs.

Among software engineering leaders, technical debt remains the most frequently cited frustration at **62%**, which indicates a persistent queue of unfinished fixes and refactors.





Security backlogs reinforce the same picture. Veracode's State of Software Security 2024 report finds that security debt exists in 42% of applications and a staggering 71% of organisations.

Even more concerning is the fact that 46% of organisations have persistent, high-severity flaws that constitute 'critical' security debt, and **48%** of discovered vulnerabilities still open after one year.

AI can increase the volume of code created, but without reliable coverage at review, testing, and remediation, risk accumulates faster than teams can address it.

Everyday friction compounds the effect. Despite - or sometimes because of - the introduction of AI, **53%** of developers still report that waiting for answers interrupts their flow, while **45%** say that knowledge silos prevent them from getting ideas across the organisation.

To date, AI has been unable to address these frictions.

Source: 2024 Developer Survey, Stack Overflow; Veracode

Why? Quick code often means less secure code

Some of the current mistrust in AI tools is justified. Independent studies document concrete security shortcomings in standard assistants and code-generation systems. In a controlled study presented at ACM CCS, participants who used an AI assistant produced code that was less secure than the control group and they were more likely to believe that their code was secure.

Separately, a large study in 2024 evaluated sixteen code-generating models and measured package hallucinations that would not resolve in real repositories. The authors reported an average hallucination rate of 22% for open-source models and 5% for commercial models, and they catalogued ~205,000 unique hallucinated package names. The same line of research and follow-up analyses describe how such hallucinations can be weaponised through slopsquatting, which is the registration of the nonexistent names to deliver malicious code.

The risks to software development and to the security of the codebase are clear:

- AI assistance can increase the likelihood that insecure patterns enter the code while developer confidence increases, which raises the chance that defects pass review
- Hallucinated dependency names expand the attack surface for supply-chain abuse through slopsquatting, which threatens the integrity of builds and releases
- Verification overhead increases because teams must confirm that recommended packages exist, are trusted, and comply with policy, which diverts effort from remediation and slows throughput

Source: Perry et al., Do Users Write More Insecure Code with AI Assistants? (ACM CCS); We Have a Package for You! A Comprehensive Analysis of Package Hallucinations by Code Generating LLMs (Spracklen et al, 2025)

22%

Average hallucination rate of open-source models (5% for commercial models)

205,474

Unique hallucinated package names across 16 LLMs in large 2024 study

> Overall, we find that participants who had access to an AI assistant wrote significantly less secure code than those without access to an assistant. Participants with access to an AI assistant were also more likely to believe they wrote secure code, suggesting that such tools may lead users to be overconfident about security flaws in their code

Perry et al., Do Users Write More Insecure Code with AI Assistants? (ACM CCS)

Why? AI tools are only being deployed in parts of the workflow, creating bottlenecks

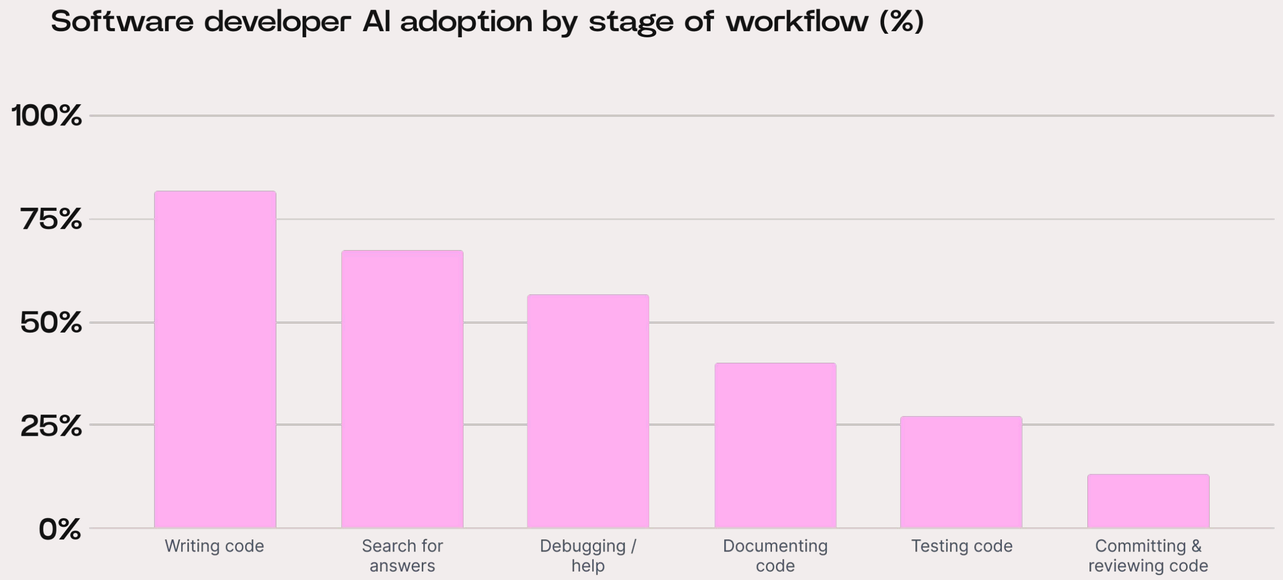
Why is AI not flowing through to overall software developer productivity?

The primary reason is that adoption is concentrated inside the editor rather than at the points where throughput is determined. In recent survey data, developers report using AI for writing code at 82%, searching for answers at 68%, and debugging at 57%. Far fewer use AI for documenting code (40%) testing (27%) or committing and reviewing code (13%). This distribution indicates an incomplete rollout that privileges early, individual tasks over the stages that decide whether changes ship.

Software delivery is governed by a sequence of gates that include commits, peer review, continuous integration, security policy, and release management. Work that is drafted with AI must still pass these gates to produce value. When assistance is absent at these stages, suggested changes often arrive without sufficient tests, without clear rationale, and without evidence of policy compliance. Reviewers respond by slowing or deferring approvals, which extends cycle times and reduces merge rates.

The result is a classic bottleneck. The volume of AI-assisted changes entering the pipeline increases, while the capacity at the review and release stages does not increase. The arrival rate rises and the service rate does not, so queues grow and backlogs persist. In this setting, higher adoption of cursor-based tools does not translate into higher organisational productivity, because the constraint lives at the gate rather than at the point of generation.

Source: 2024 Developer Survey, Stack Overflow



Why? Incomplete rollout harms the trustworthiness and contextual accuracy of AI

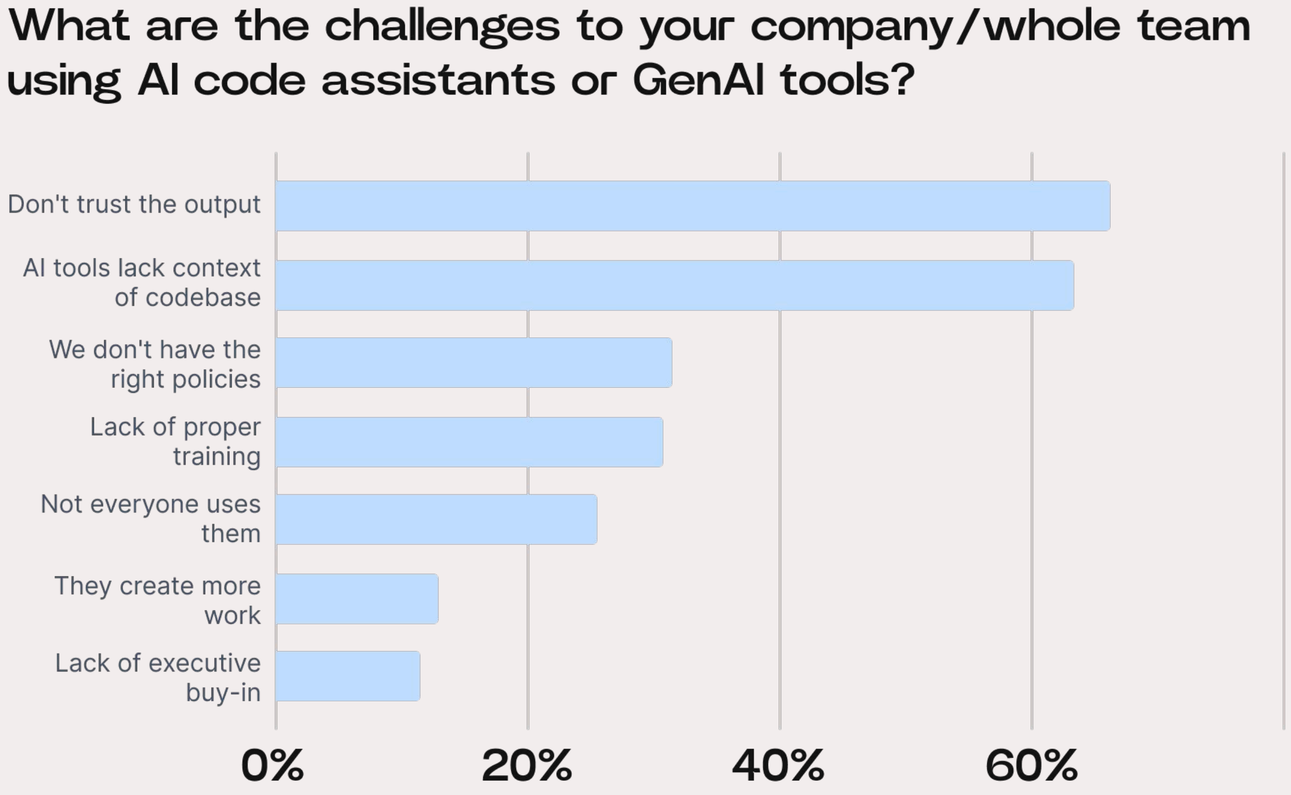
Incomplete rollout also reduces perceived accuracy at the gate.

Reviewers evaluate AI-generated changes when those changes reach tests, peer review, security checks, and release management. When AI is deployed mainly inside the editor, reviewers often receive proposals without unit tests, without policy results, and without a clear rationale tied to the repository. In that setting, confidence falls and approval rates slow because the evidence required for a safe decision is missing.

Survey data is consistent with this pattern. The most frequently cited blockers are trust and context rather than training or executive sponsorship. **66%** of respondents report that they do not trust the output of AI tools, and **63%** report that the tools lack codebase context. Additional concerns such as gaps in policy and uneven adoption appear lower in the ranking and do not explain the sharp drop in confidence.

Accuracy is therefore assessed as a property of the surrounding system rather than as a property of the model. When reviewers see tests passing, policies satisfied, and an explanation that links the change to the relevant files, they are more likely to accept the result as accurate. When those elements are absent, the same code is judged as unreliable even if the underlying model is unchanged.

Source: 2024 Developer Survey, Stack Overflow



Why not to lose hope | There are clear evidenced benefits of AI when used well

+9%
PRs per developer

+15%
Merge rate

+84%
Successful builds

↓67%
Reduction in median time from detection to remediation

Source: GitHub

AI still delivers measurable benefits when it operates through the delivery pipeline. Controlled studies show that assistants can accelerate individual tasks. In Accenture's enterprise-scale rollout, GitHub reports 9% more pull requests per developer, a 15% higher merge rate, and an 84% increase in successful builds. These results indicate that more changes are both produced and accepted by reviewers and automation.

Security operations show a similar pattern. During its public beta, Copilot Autofix reduced the median time from detection to remediation by about 67%. This outcome demonstrates that speed and safety can improve together when changes carry tests, policy checks, and clear rationale through review and continuous integration.

The practical conclusion is straightforward: **speed is an outcome of orchestration.** When AI-touched changes arrive with evidence that satisfies tests and policy, they move faster, they merge more often, and they remain stable once released.

The solution | Let custom-built AI fully orchestrate the developer workflow

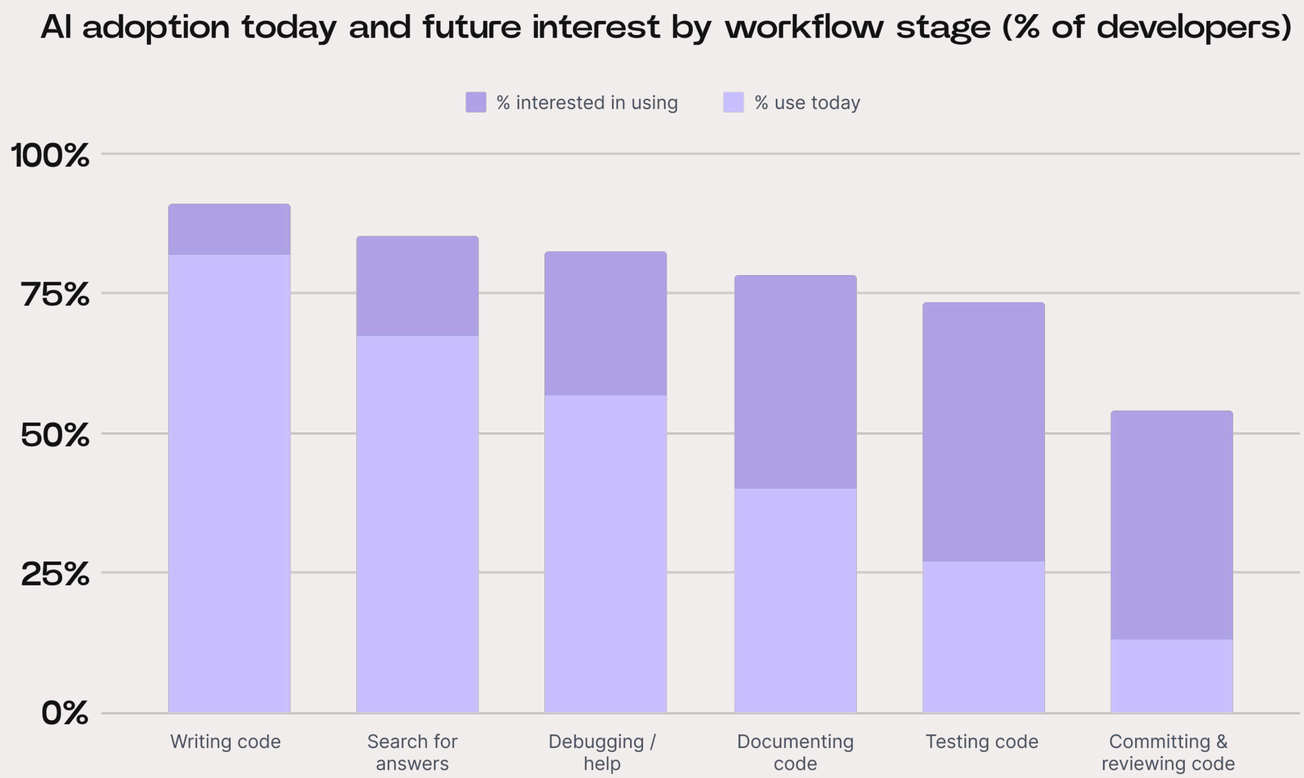
So what is the solution? Let custom-built, best-in-class AI, built specifically for coding contexts, take a broader role across the delivery pipeline so that it contributes not only to drafting but also to submission, verification, and release.

Today the rollout is concentrated inside the editor. That pattern improves typing speed but does not raise throughput at the gates where decisions are made. Reviewers, continuous integration, security policy, and release management remain the determining stages, and they are often under-served by current deployments.

The corrective action is to extend coverage at the gate. Policy-aware agents should open pull requests with diffs and tests, run continuous integration, satisfy security policy, and attach audit trails so that reviewers can approve changes with confidence. When evidence accompanies the change, approvals accelerate and merge rates improve.

Interest data indicates that teams are ready to expand coverage. **41%** of respondents want AI to assist with commit and review, **46%** want AI to assist with testing, and **40%** want AI to assist with deployment and monitoring. Platform signals point in the same direction. **67%** report that their software development life cycle is mostly or completely automated, and **64%** want to consolidate toolchains. These figures indicate demand for orchestration rather than additional isolated tools.

The recommendation is to instrument AI rather than retreat from it. Require tests, policy results, and dependency verification on every AI-touched change, and present the evidence in the pull request. When AI operates through the pipeline with proof, security becomes a speed feature and the benefits are visible in throughput metrics.

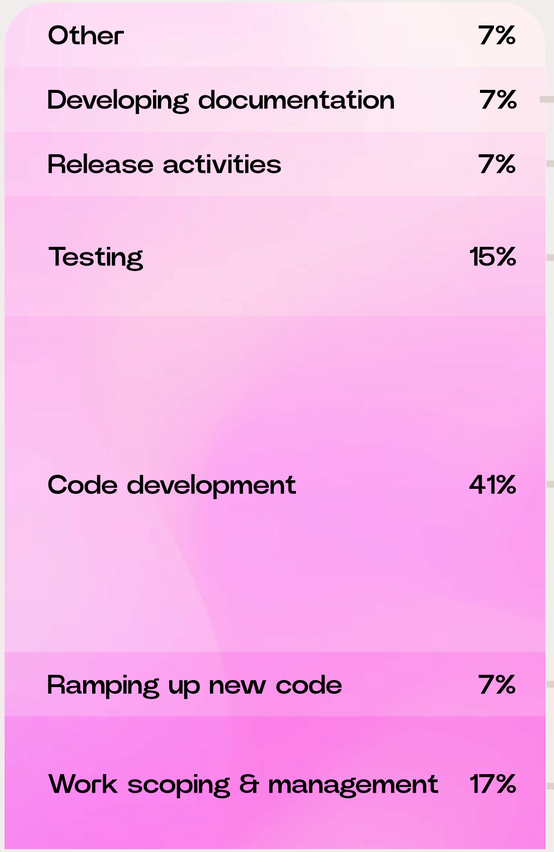


Source: 2024 Developer Survey, Stack Overflow

What do winners do differently? | Don't stop at "AI in the IDE"

The highest-performing teams don't stop at "AI in the IDE." **They operate AI through the pipeline** - from PR creation to CI and policy - so changes arrive with evidence and clear ownership.

The best AI tools touch every element of developer activity:



Developing documentation: AI can generate and continuously update architecture, API, and run-book documentation from code, tickets, and commits, linking each document to the relevant pull requests to preserve traceability.

Release activities: AI can assemble release notes and changelogs from merged pull requests, verify policy and license compliance, stage deployments after green builds, and schedule or roll back releases according to predefined criteria.

Testing: AI can propose and maintain unit, integration, and regression tests, expand coverage in high-risk areas identified from incident history, stabilise flaky tests, and attach passing evidence to each pull request.

Code development: AI can execute small, high-volume changes such as refactors, lint fixes, and dependency updates, produce minimal diffs with clear rationale, and ensure conformance with security policies and coding standards before submission.

Ramping up new code: AI can prepare onboarding briefs that explain system boundaries, data models, and coding patterns, generate code tours for key modules, and surface representative examples to accelerate first contributions.

Work scoping and management: AI can triage and de-duplicate backlog items, cluster related work into coherent packages, estimate effort from historical data, propose owners based on expertise, and produce short execution plans with risks and acceptance criteria.

Source: "Beyond Code Generation: More Efficient Software Development", Bain & Company

% of developer working time spent per activity

What next? | The 90-day plan for tech leaders

Objective: Extend AI from the editor to the delivery gates so that changes arrive with proof and move through review, continuous integration, security policy, and release.

Weeks 1 to 2

Foundation

- Appoint a single accountable owner for AI in engineering and name platform, security, and enablement leads.
- Baseline the flow by recording pull requests per developer, merge rate, time to merge, build success, time to remediation, and aged issues.
- Select two high volume flows such as test scaffolding and small bug fixes and documentation.
- Decide the execution boundary and prefer virtual private cloud or on premises for sensitive work.
- Publish a reviewer checklist that requires tests, policy results, dependency verification, and a short rationale with every AI touched change.

Weeks 3 to 6

Pilot

- Stand up the agent runtime and connect repositories, issue tracking, continuous integration, and code scanning.
- Require evidence rich pull requests from the pilot agents and enforce policy as code in continuous integration.
  Run the pilot with two squads and review telemetry each day.
  Hold a weekly risk review with security to sample AI touched changes and confirm audit trails.
- Report interim results at the end of week 6 and decide whether the pilot is meeting targets.

Weeks 7 to 12

Expand & decide

- Extend coverage to dependency updates, flaky test repair, and security remediation with suggested fixes.
- Add two to four more squads and keep daily telemetry and weekly governance reviews.
  Publish a single dashboard for leadership that shows outcomes by team and repository.
- Decide on scale out at the end of week twelve and confirm budget, ownership, and service levels.

Success

- Every AI touched change includes passing tests, clean security and dependency checks, and a concise explanation of intent and impact.
- All evidence is attached to the pull request and stored with an audit trail.

Targets

- Pull requests per developer: ↑ 15%
- Merge rate: ↑ 15%
- Time to merge: ↓ 30%
- Build success: ↑ 30%
- Time to remediation: ↓ 60%
- Aged issues: ↓ 30%

Closing thoughts

AI adoption across engineering is high, yet impact is still constrained because throughput is decided at the gates. Commit and review, continuous integration, and policy are where trust is earned or lost. IDE assistants help teams type faster. They do not, by themselves, help teams ship faster.

The core bottleneck is incomplete rollout. Most teams deploy AI inside the editor but not at the gates, so commit and review remain under-served where it matters most. Suggestions enter the pipeline in greater volume, but too few arrive at the gate with the proof that reviewers require.

Speed only counts when it reaches main. Leaders should measure pull requests per developer, merge rate, time to merge, build success, escaped defects, and time to remediation rather than keystrokes or suggestion counts. Security should be treated as a speed feature by encoding it in the pipeline with policy as code, scanning, and autofix so that cycle time falls while risk declines.

The scalable pattern is to operate AI as a pipeline participant. Policy-aware agents should open evidence-rich pull requests, run tests and scans, satisfy policy, and present a concise rationale to reviewers. Reviewers grant approval faster when every AI-touched pull request arrives with proof, including passing tests, clean scans, policy compliance, and an explainable diff. Trust then becomes a property of the system rather than of the model alone.

The quickest path to felt results is to start narrow with two high-volume flows such as test scaffolding and small fixes or documentation. Run them inside your boundary in a virtual private cloud or on premises to compress approvals and simplify compliance. Establish a single source of truth by standing up a dashboard as code and reviewing it each week with pull request, continuous integration, and security telemetry.

Reduce tool sprawl in favour of orchestration so that process seams and context gaps shrink and AI can carry changes through the gate rather than stopping at the cursor. The north star is simple. Throughput equals adoption multiplied by gate coverage and evidence quality. Improving any one of these factors multiplies overall impact.

For high-performing tech leaders, the best next step is to run a ninety-day pilot that proves merge rate up, time to merge down, builds green up, and aged debt down, and then to scale to dependency updates and security remediation once the improvements are verified.


COSINE