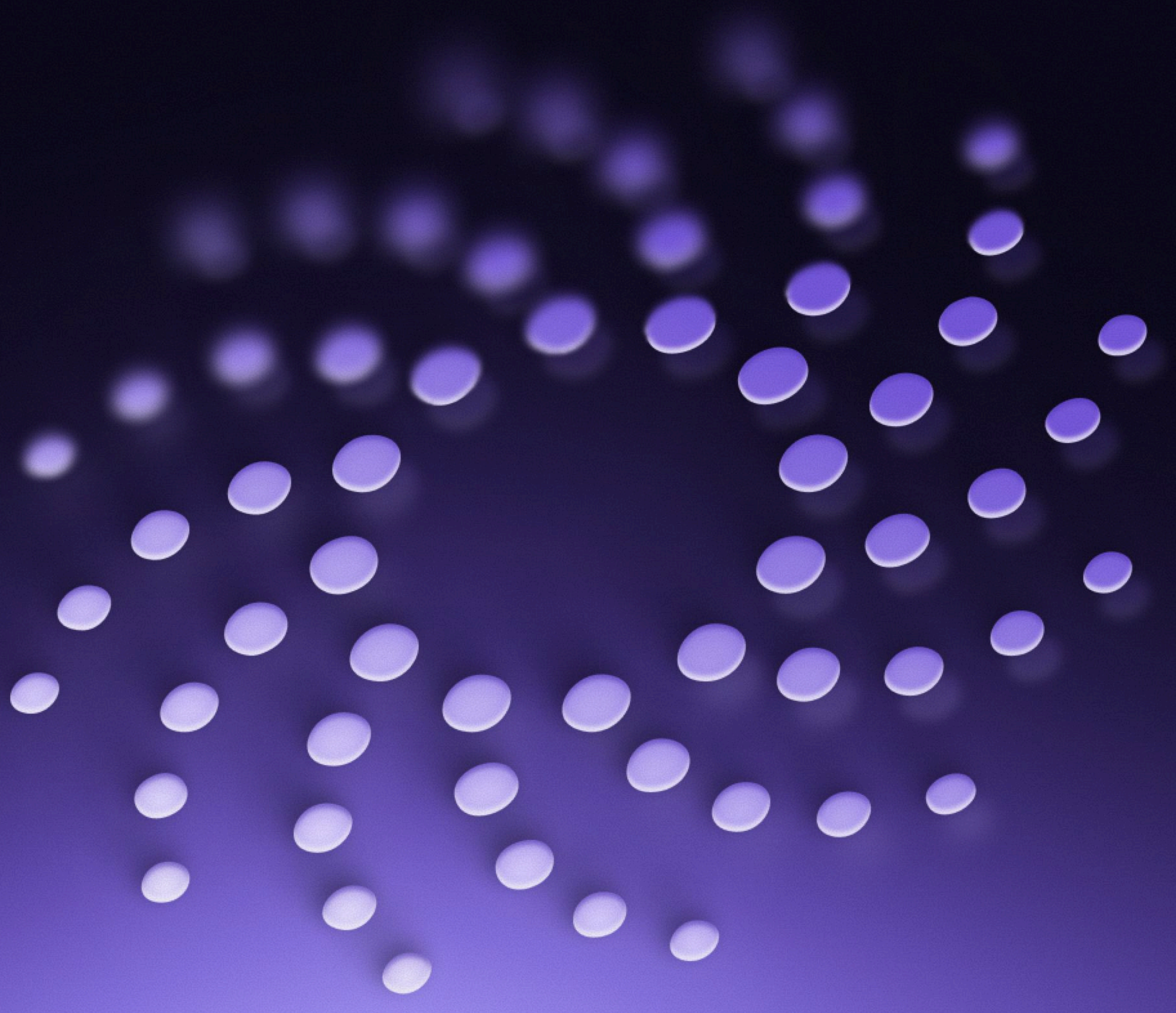
cortex

Engineering in the Age of AI

2026 Benchmark Report

How engineering leaders are navigating the AI revolution — and what they're learning about the hidden costs of speed



cortex.io

Executive summary

Engineering teams are moving faster than ever. AI coding assistants have transformed how developers write code, with organizations reporting dramatic increases in development velocity.

However, this speed comes at a steep cost. Although developers are shipping more code and doing so faster than ever, engineering leaders are discovering quality and stability problems they didn't see coming.

We surveyed over 50 leaders and analyzed development metrics across multiple organizations. What we found reveals a critical gap between AI adoption and the engineering foundations needed to support it safely. It's never been clearer that AI amplifies both the best and worst aspects of your engineering practices.

Key findings

Development velocity is up dramatically

PRs per author increased by 20% year-over-year, and most teams are deploying more frequently.

Quality challenges are emerging

Incidents per pull request increased by 23.5%, with higher change failure rates and longer resolution times as code volume grows.

Governance remains inconsistent

Only 45% of organizations have formal AI usage policies.

Foundations are the differentiator

Teams with strong service ownership and testing practices see better AI outcomes.

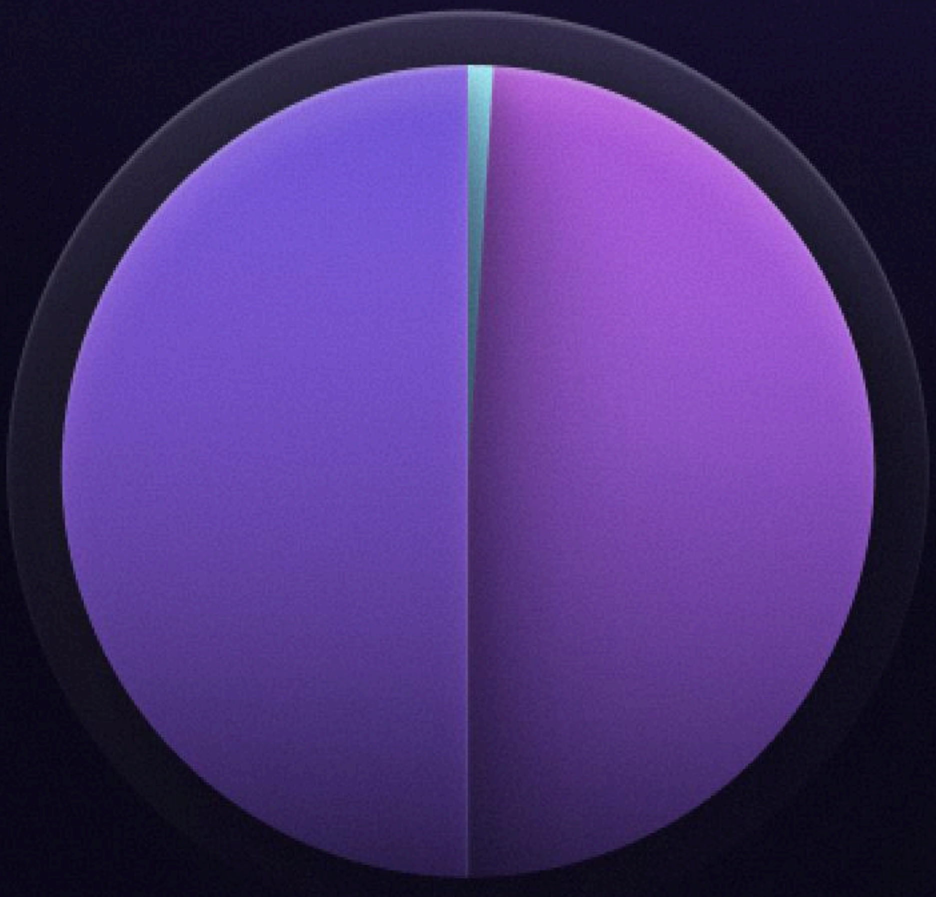
Engineering leaders need to be intentional about AI adoption, not just enthusiastic. Organizations that thrive in 2026 will invest in engineering foundations now while managing the pace of AI integration carefully.

Everyone's using AI; but not the same way

90% adoption, zero standardization

After a brief period of "early" experimentation with AI, coding assistants have quickly become standard tools across engineering organizations of all sizes. Nearly 90% of engineering leaders report their teams actively use AI tools, with usage ranging from individual experimentation to mandatory adoption across all teams.

Current adoption levels:

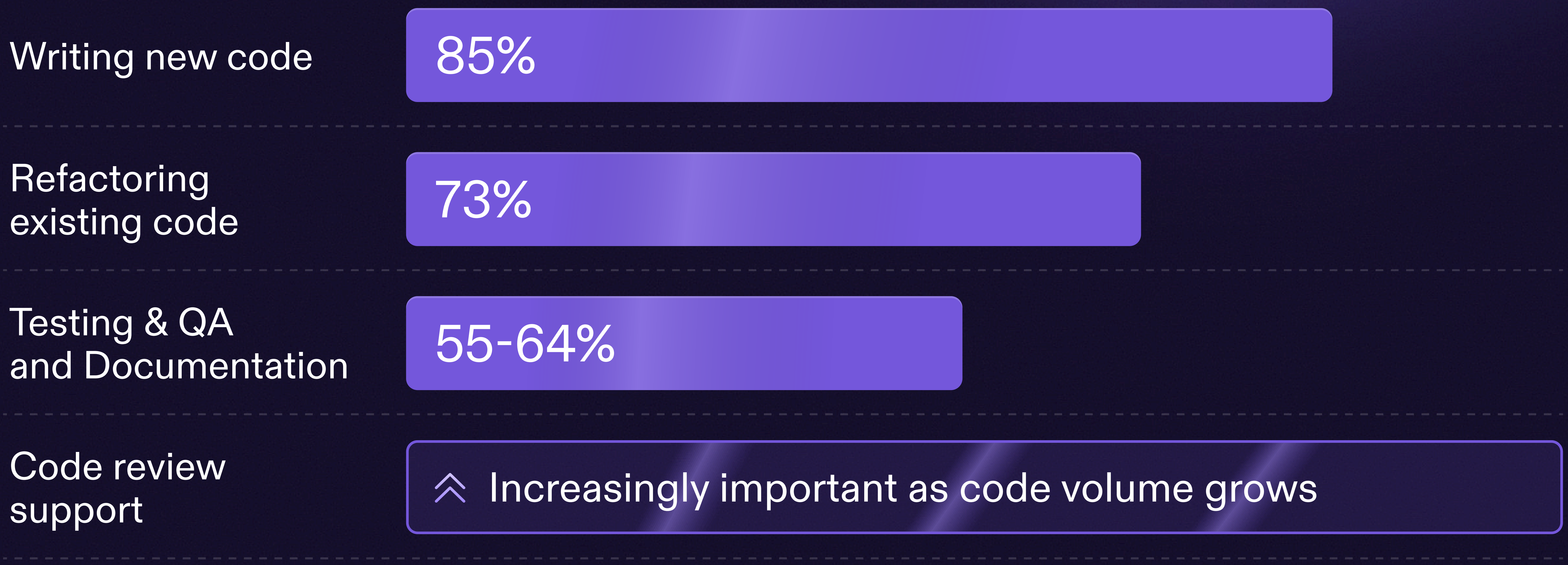


● **50%** Widely adopted across teams

● **49%** Selectively adopted for specific use cases

● **>1%** Pilot/exploration phase

The most common use cases remain focused on developer productivity:



But a more sophisticated pattern is emerging. Some teams are moving beyond simple code completion and starting to wire together entire development workflows. A designer finishes a component in Figma, and an AI agent picks up that design, understands the organization's established patterns and coding standards, and generates a production-ready UI component.

AI isn't just helping developers write code anymore. It's becoming woven into the development process itself, acting as a partner that understands context, follows organizational conventions, and bridges the gap between design and implementation.

A patchwork of tools, a mess of results

Despite widespread adoption, most organizations are struggling with standardization. The reality is that most teams are operating with a patchwork of AI tools. Some are using GitHub Copilot, others experimenting with Cursor or Claude, and many relying on individual subscriptions to various AI services.

This tool fragmentation leads to context fragmentation, a critical operational gap. When each AI assistant operates with a different model of the company's architecture and standards, their suggestions become actively counterproductive. An AI will generate code that appears correct in isolation but breaks established patterns or creates hidden dependencies.

The result is that developers on the same team receive contradictory outputs for the same task. Without a unified context, one engineer's AI-generated component might be production-ready while others ignore the company's design system entirely, creating unnecessary rework and frustration.

This fragmentation creates several problems that engineering leaders are only beginning to grapple with. Teams using different AI tools produce inconsistent code quality. Security risks multiply when there's no centralized governance over which tools can access sensitive codebases. Most critically, measurement becomes nearly impossible when every team adopts AI differently.

The standardization challenge extends beyond just tooling. Organizations need to standardize on usage patterns, review processes, and quality standards. Without this standardization, AI adoption creates more complexity than it solves.

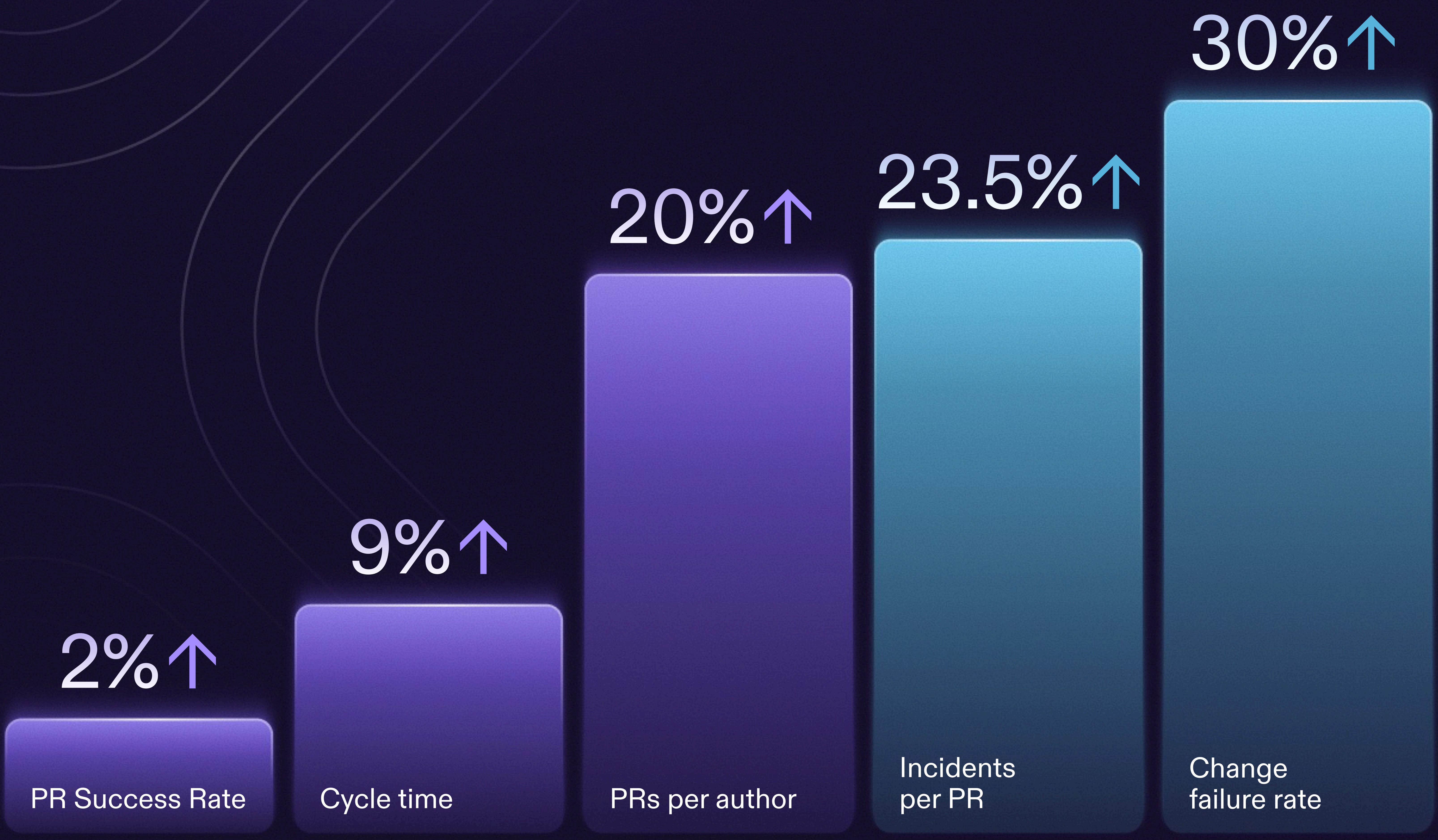
The productivity paradox

Speed at a cost

The metrics tell a compelling and concerning story about AI's impact on development velocity. Our analysis of development metrics across multiple organizations reveals significant increases in output, but also in complexity.

Key performance changes

Year-over-year, Q3 2024 to Q4 2025



These numbers reveal what we're calling the "Productivity Paradox": AI makes developers faster, but the increased velocity creates downstream challenges that erode the gains. Developers ship more code, but that code introduces more bugs and stability issues. For every pull request merged, teams are seeing nearly a quarter more incidents compared to a year ago.

cortex

More code, more bugs

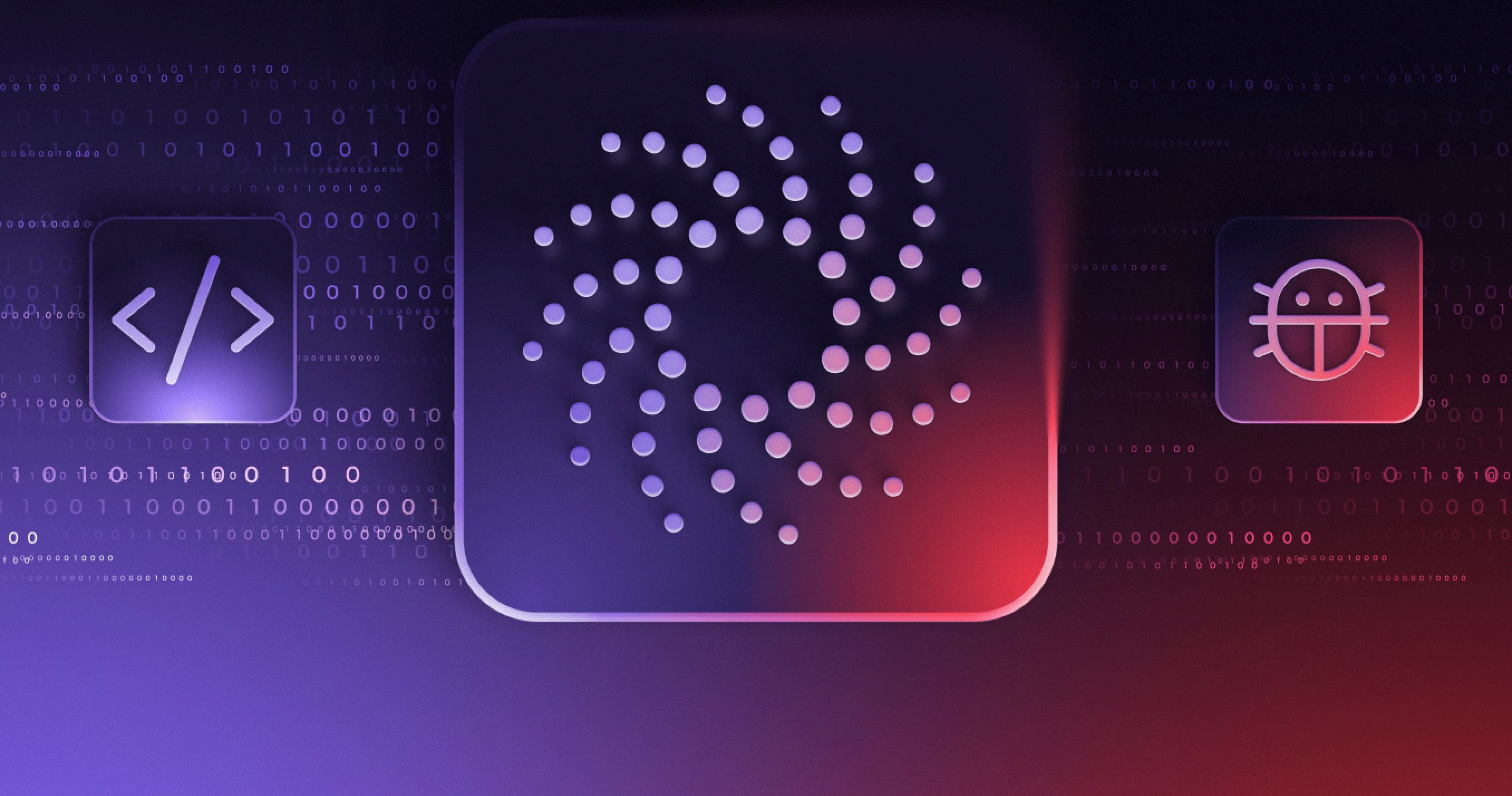
The change failure rate increase of 30% reveals that while AI helps developers write more code faster, it also introduces more bugs into production. The increased velocity comes with a quality trade-off that many organizations weren't prepared for.

Traditional code review processes are struggling to keep pace with AI-generated code volume. Reviewers are spending more time on each pull request, and the pressure to maintain velocity often leads to rushed reviews. The result is a higher rate of bugs making it to production.

Some organizations adapt by implementing enhanced review requirements, using AI-assisted reviews, or adding quality gates that prevent AI-generated code from reaching production without thorough human review. However, maintaining quality becomes exponentially harder when the volume of code being generated increases dramatically.

Not everything is trending downward, however: PR success rate increased by 2%, suggesting that while developers are shipping more code, they're getting slightly better at getting their changes merged on the first attempt. This modest improvement, however, is overshadowed by the downstream quality impacts once code reaches production.

Confidence in AI's impact remains high despite these challenges. Among our respondents, 33% report being "very confident with data to prove it" that AI is improving outcomes, while another 58% are "somewhat confident but mostly anecdotal." When it comes to ROI, 25% report clear positive returns, while 50% believe ROI is "likely positive but not yet quantified." This gap between confidence and measurement underscores how most organizations believe AI is working, but few have the data infrastructure to prove it conclusively. This will likely change in 2026, as leaders shift focus to proving ROI with data rather than confidence alone.

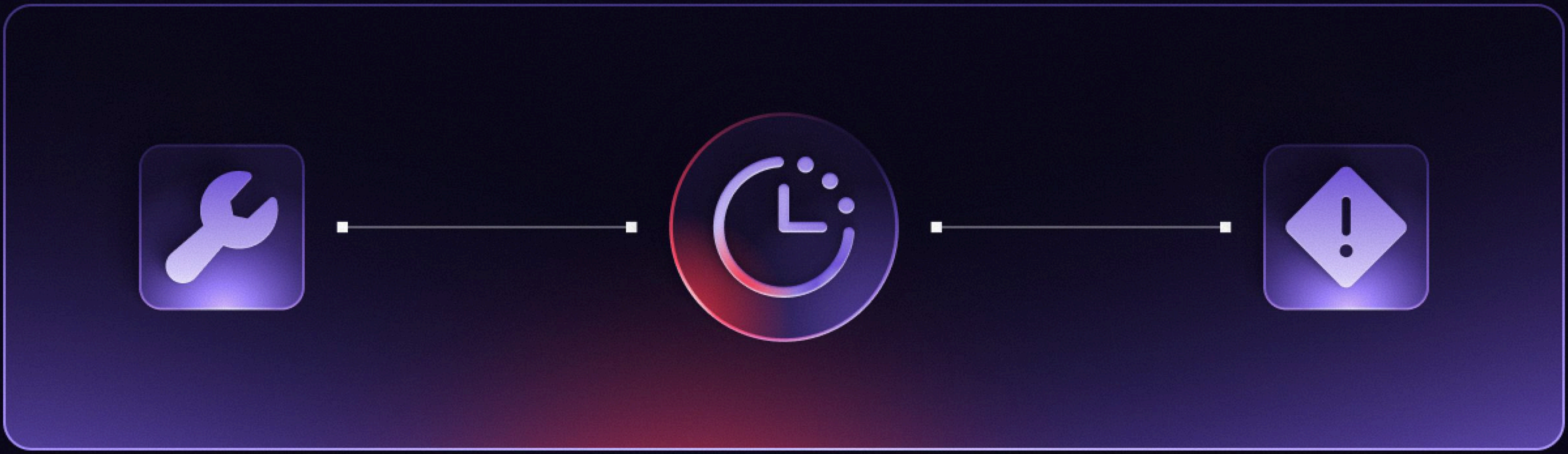


When things break, recovery is a longer process

Engineering leaders report that resolution times are increasing as AI-generated code becomes more prevalent. When systems fail, teams are taking longer to restore service compared to before widespread AI adoption.

Several factors contribute to this challenge. The increased complexity of AI-generated code makes debugging harder. Engineers responding to problems often encounter code they didn't write and don't fully understand, even if they or their teammates originally submitted it. The higher volume of changes and deployments creates more potential failure points, making root cause analysis more time-consuming. On-call teams are also dealing with more frequent interruptions, leading to fatigue and slower response times.

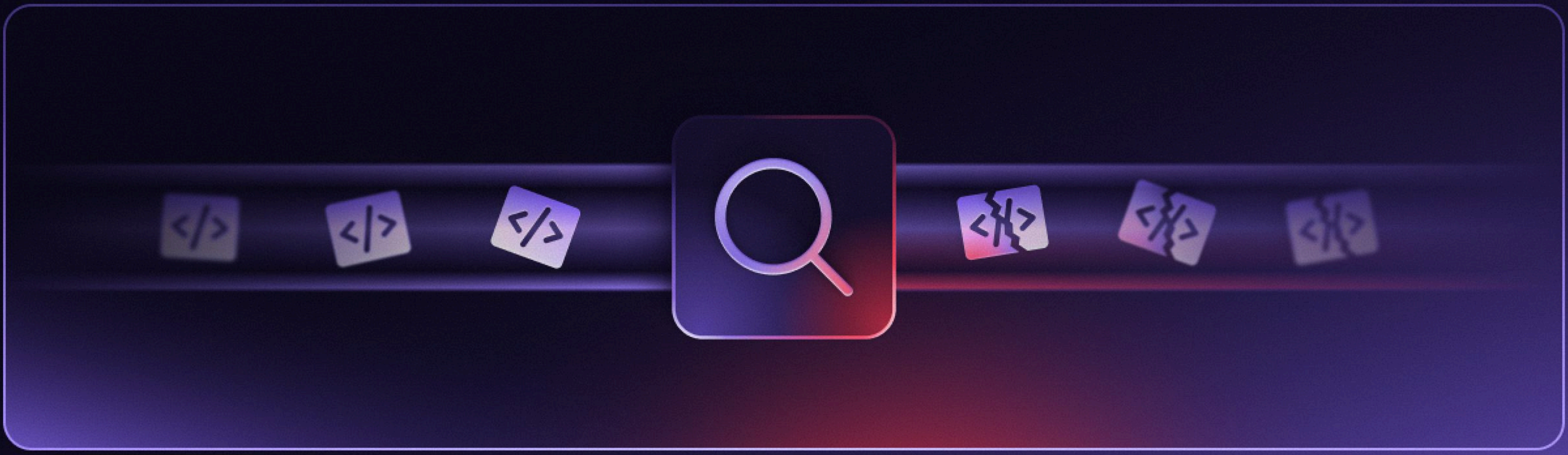
For engineering leaders, longer resolution times represent significant business risk. The cost in lost revenue, customer trust, and team burnout compounds with each problem. Organizations optimizing for velocity without investing in response capabilities are building systems that fail faster and recover slower.



Code review can't keep up

The review process is evolving in response to AI-generated code, but not fast enough. Organizations are experimenting with different approaches, from requiring multiple reviewers for AI-generated code to implementing automated quality checks that flag potential issues.

However, many engineering leaders report that their review processes weren't designed for this volume of code. Traditional approaches assume that developers have a deep understanding of the code they're submitting. With AI-generated code, that assumption breaks down. Reviewers need to understand not just what the code does, but how the AI arrived at that solution and whether it's appropriate for the context.

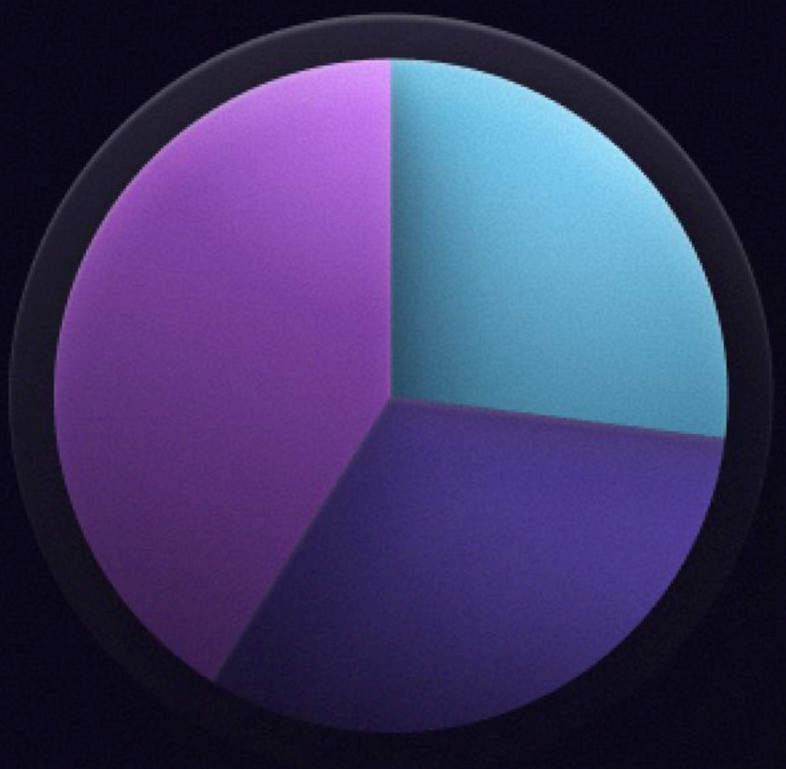


Who's watching the AI?

The policy vacuum

Despite widespread AI adoption, governance remains inconsistent across organizations. Our survey reveals that only 32% of engineering leaders report having formal AI usage policies, while another 41% rely on informal guidelines, and 27% operate without formal governance.

Current governance approaches:



● 32% Formal policies with enforcement

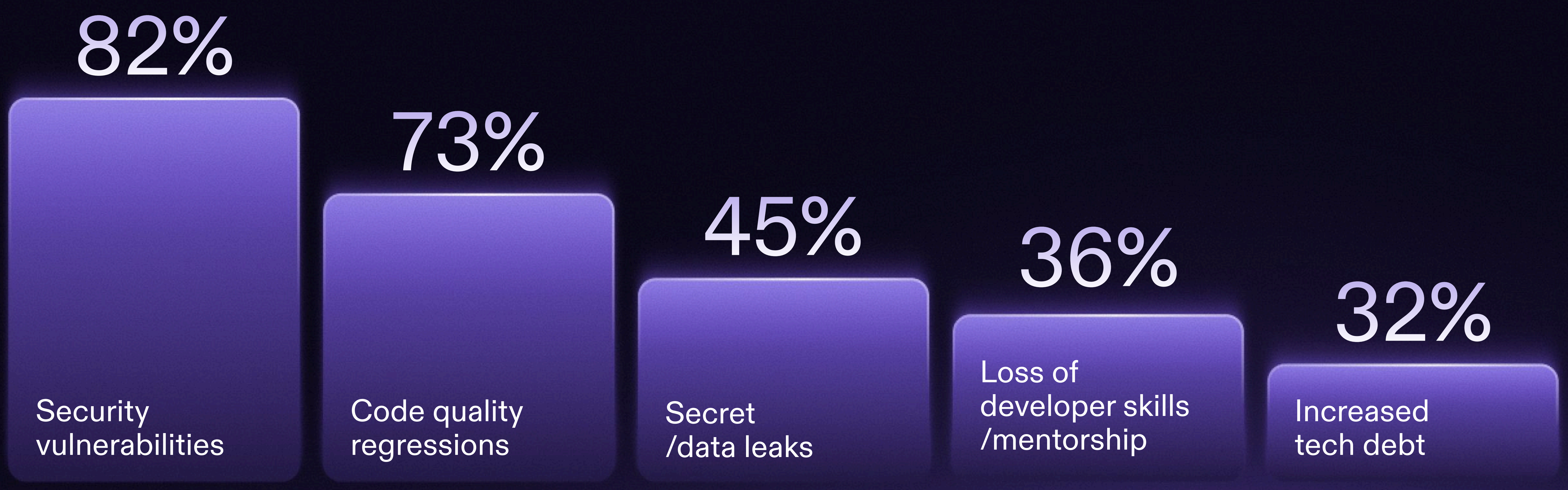
● 41% Informal guidelines

● 27% No formal governance

This gap is particularly concerning given the security and compliance risks associated with AI tools. Organizations essentially fly blind when it comes to understanding how AI tools are used, what data they access, and what risks they introduce.

What could go wrong?

The top concerns about AI-assisted code are consistent across organizations include:



These concerns are driving different governance approaches. Some organizations are taking a restrictive stance, banning AI tools from accessing sensitive codebases entirely. Others are implementing comprehensive policies that govern which tools can be used, how they can be used, and what data they can access.

The challenge is that traditional security policies weren't designed for AI tools that can access vast amounts of code and potentially leak sensitive information. Organizations need new approaches that balance productivity gains with security requirements.

Are developers forgetting how to write code?

A particularly interesting concern is the "loss of developer skills/mentorship." Many engineering leaders worry that over-reliance on AI could erode fundamental engineering capabilities. The concern isn't just about individual developers losing skills, it's about organizations losing institutional knowledge and the ability to maintain and evolve their codebases.

This suggests that successful AI adoption requires not just technical governance, but also investment in developer education and skill development. Organizations need to ensure that developers understand the code they're submitting, even when it's AI-generated, and that they maintain the critical thinking skills needed to evaluate AI suggestions.

The more significant risk is the loss of system context. When developers rely on AI to generate code, they bypass the process of learning a system’s history, trade-offs, and constraints. This unwritten knowledge, the "why" behind the "what," is essential for debugging complex outages and planning long-term architectural changes.

Shipping code without this deep context ultimately builds a brittle system. It may function correctly today, but it becomes nearly impossible to safely evolve. Over time, the organization loses its ability to produce future system experts who can confidently own and improve critical services.

What separates the winners from the losers

Strong foundations amplify AI's strengths

Our research reveals that organizations with strong engineering foundations see better outcomes from AI adoption. Engineering leaders consistently identify these as the most critical foundations:

Testing & automation coverage, secure coding practices, and clear ownership

Top tier priorities, each mentioned by 30-40% of respondents

Service catalogs and comprehensive documentation

Critical for providing AI tools with context

Incident response practices

Increasingly important as code volume grows

Vulnerability management

Essential for securing the increased volume of AI-generated code

What's particularly notable is how evenly distributed the top priorities are, suggesting that successful AI adoption requires balanced investment across multiple foundational areas rather than excelling in just one.

These foundations aren't just a nice-to-have when using AI, they're essential. Organizations with weak foundations see AI amplify their existing problems. Those with strong foundations see AI amplify their strengths.

Given the challenges with resolution times and increase in both incidents and change failure rate, incident response practices deserve special attention. Organizations need robust runbooks, clear escalation paths, comprehensive monitoring, and well-documented services. When AI-generated code fails in production, responders need context to diagnose and fix issues quickly. Without strong incident response foundations, the cost of increased velocity becomes unsustainable.

Why ownership matters more than ever

Clear service ownership emerges as a particularly critical foundation for AI success. Organizations with well-defined service boundaries and ownership models report better outcomes from AI adoption.

This makes sense: when AI tools have clear context about service responsibilities, they generate more appropriate code. When ownership is unclear, AI-generated code often crosses service boundaries inappropriately or creates dependencies that weren't intended.

The challenge is that many organizations adopted AI tools before establishing clear service ownership. They're now trying to retrofit governance onto AI adoption, which is much harder than building governance into the adoption process from the beginning.

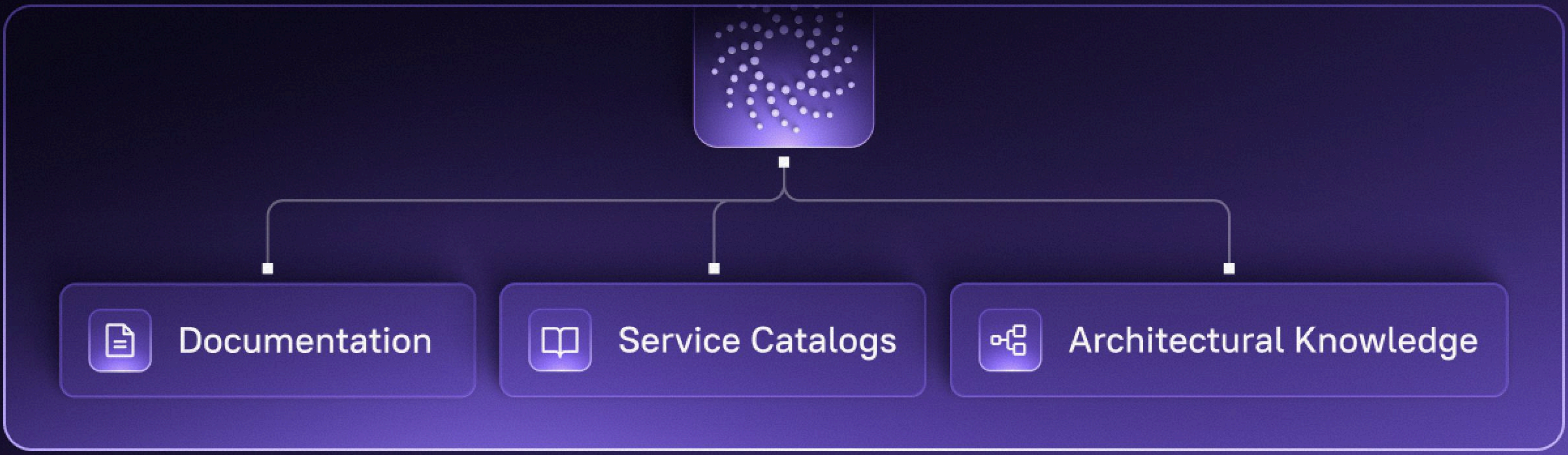
Public Subnet
Backend team
12 members

AI needs context to write quality code

The importance of documentation for AI success cannot be overstated. AI tools need context to generate appropriate code, and that context comes from documentation, service catalogs, and architectural knowledge.

Organizations with comprehensive documentation find that AI tools generate more appropriate code. Those with sparse documentation find that AI tools generate code that doesn't fit the existing architecture or follows patterns inconsistent with the rest of the codebase.

The challenge is that documentation is often the first thing to fall by the wayside when teams are under pressure to deliver features quickly. AI adoption makes this problem worse, as teams generate more code faster but don't have time to document it properly.



What's next

Four shifts on the horizon

Based on our research, several trends are emerging that will shape AI adoption in 2026.

01 Measuring AI's actual impact

Engineering leaders will shift focus from a blind faith that AI works to proving it with data. Organizations that can quantify how AI is changing their teams will have a significant advantage in making strategic decisions about adoption, investment, and governance.

02 Shift from individual tools to integrated platforms

Organizations move beyond individual AI coding assistants toward integrated platforms that combine coding assistance with governance, testing, and deployment capabilities. The goal is to create a unified development environment that provides AI assistance while maintaining quality and security standards.

03 Specialized AI tools

There's growing demand for AI tools specialized for specific tasks rather than general-purpose coding assistants. Organizations want AI tools that can generate test cases, create CRUD operations from database schemas, or build UI components from design specifications, rather than trying to do everything with a single general-purpose tool.

04 Enhanced governance automation

Organizations develop automated governance systems that can monitor AI usage, enforce policies, and provide real-time guidance to developers. The goal is to make governance seamless rather than burdensome.

What engineering leaders should do next

Based on our research, here are strategic considerations for engineering leaders planning their 2026 AI approach.

Consider your engineering foundations

Before scaling AI adoption, evaluate your current engineering foundations: comprehensive testing, clear service ownership, robust documentation, and effective incident response processes. AI amplifies whatever you have, good or bad. The growing challenges with resolution times underscore why response capabilities must be strengthened before, not after, accelerating development velocity.

Prioritize incident response readiness

Given the increase in incident resolution times, evaluate your incident response foundations now. Ensure services have comprehensive runbooks, monitoring is in place, escalation paths are clear, and responders have the context they need to debug AI-generated code quickly. The cost of critical incidents quadrupling in duration can outweigh the benefits of increased development speed.

Evaluate your governance needs

Consider developing formal AI usage policies that address security, quality, and compliance requirements. Many organizations find automated monitoring and enforcement mechanisms help scale AI adoption safely.

Assess tooling standardization

Consider moving beyond ad-hoc AI tool adoption toward standardized, organization-wide tooling with consistent policies and training. Productivity gains from AI are often maximized when teams use the same tools consistently.

Consider developer education needs

Evaluate whether your developers need training to use AI tools effectively while maintaining code quality and understanding. The most successful organizations ensure developers understand both the capabilities and limitations of AI tools, and can debug the code they generate.

Define meaningful metrics

Consider implementing metrics that track both velocity improvements and quality outcomes from AI adoption. Track not just deployment frequency and PR volume, but also change failure rates, incident volumes, and MTTR, especially for critical incidents. These lagging indicators reveal the true cost of velocity gains.

Learn how Vista powers AI-driven engineering with Cortex. vistaprint.

Humans and AI aren't replacing each other

The most successful organizations view AI not as a replacement for human developers, but as an amplifier of human capabilities. The future belongs to teams that effectively combine human creativity and problem-solving with AI's speed and pattern recognition.

The next generation of AI tools will be more "magical," which is a whimsical way of saying they'll do more work automatically while requiring less human oversight. But this will only be possible in organizations that have invested in the foundations that make AI safe and effective. The data makes it clear that organizations that prioritize velocity over foundations are creating systems that ship faster but encounter more quality and stability challenges when issues arise.

Organizations that thrive in 2026 will be intentional about AI adoption, investing in testing, documentation, service ownership, and especially incident response capabilities, while maintaining the human judgment and creativity that remain irreplaceable. The goal isn't to slow down, it's to build the foundations that make sustainable speed possible.

Methodology

This report is based on:

- Survey responses from 50 engineering leaders across organizations of various sizes
- Analysis of development metrics from multiple organizations
- Additional survey data from engineering professionals at IDPCON
- Internal research and analysis

Key metric definitions

Throughout this report, we reference several engineering metrics to quantify AI's impact on development velocity and quality. Here's what each metric measures:

Development velocity metrics:

Merged PRs per author

The average number of pull requests each developer completes and merges over a given period. This measures productivity in the context of how much code made it into the codebase. While higher numbers might indicate increased output, this should be evaluated alongside other quality metrics.

Cycle time

The time from when a developer starts work on a PR to when it's merged. This measures how quickly code moves through the coding, review, and approval process, including the time spent in development and iterating. Lower cycle times indicate efficient code review workflows, though this metric currently doesn't capture the time from merge to production deployment.

PR success rate

The percentage of pull requests that are successfully merged out of all pull requests that have been completed (merged or closed without merging). Higher rates suggest developers are submitting appropriate, aligned changes rather than work that gets abandoned.

Deployment frequency

The number of times code changes are deployed to production for each service over a given period. This can be measured at the service, developer, or organizational level. Higher deployment frequency typically correlates with smaller, less risky changes and indicates a team's ability to deliver value continuously.

Quality and reliability metrics:

Change failure rate

The percentage of deployments that result in a rollback. Calculated as deployments marked with "ROLLBACK" status divided by total deployments. This is a core DORA metric that measures deployment quality. This metric currently doesn’t capture hotfixes or non-rollback failures.

Incident frequency (or incidents per service)

The total number of incidents opened for a service over a given period. This measures how often a service experiences problems that require intervention. Rising incident frequency often signals quality issues, insufficient testing, or increased system complexity.

MTTR (Mean Time to Recovery)

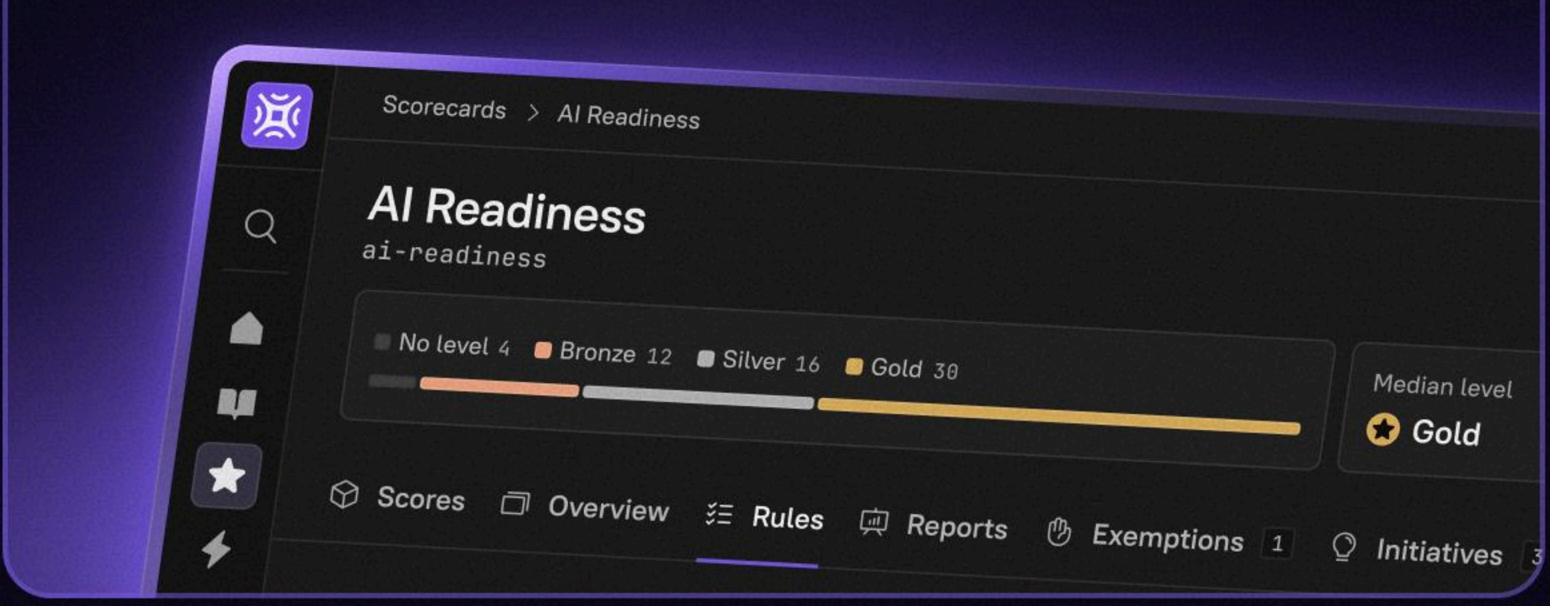
The average time from when an incident is triggered to when it's marked as resolved. This measures how quickly teams can detect, diagnose, and fix production issues. Lower MTTR indicates strong incident response capabilities and effective troubleshooting processes.

These metrics provide a baseline for understanding how AI adoption affects both the speed at which teams deliver code and the quality of what they ship. The goal isn't to hit specific benchmarks, it's to understand trends over time and take action based on what the data reveals about your team's performance.

About Cortex

Cortex is the AI-powered Internal Developer Portal that helps engineering leaders build organizations that ship reliable, secure, and efficient software, faster. It uses AI to make sense of your engineering data, figure out the next unlock for your team, and solve it, all in one place.

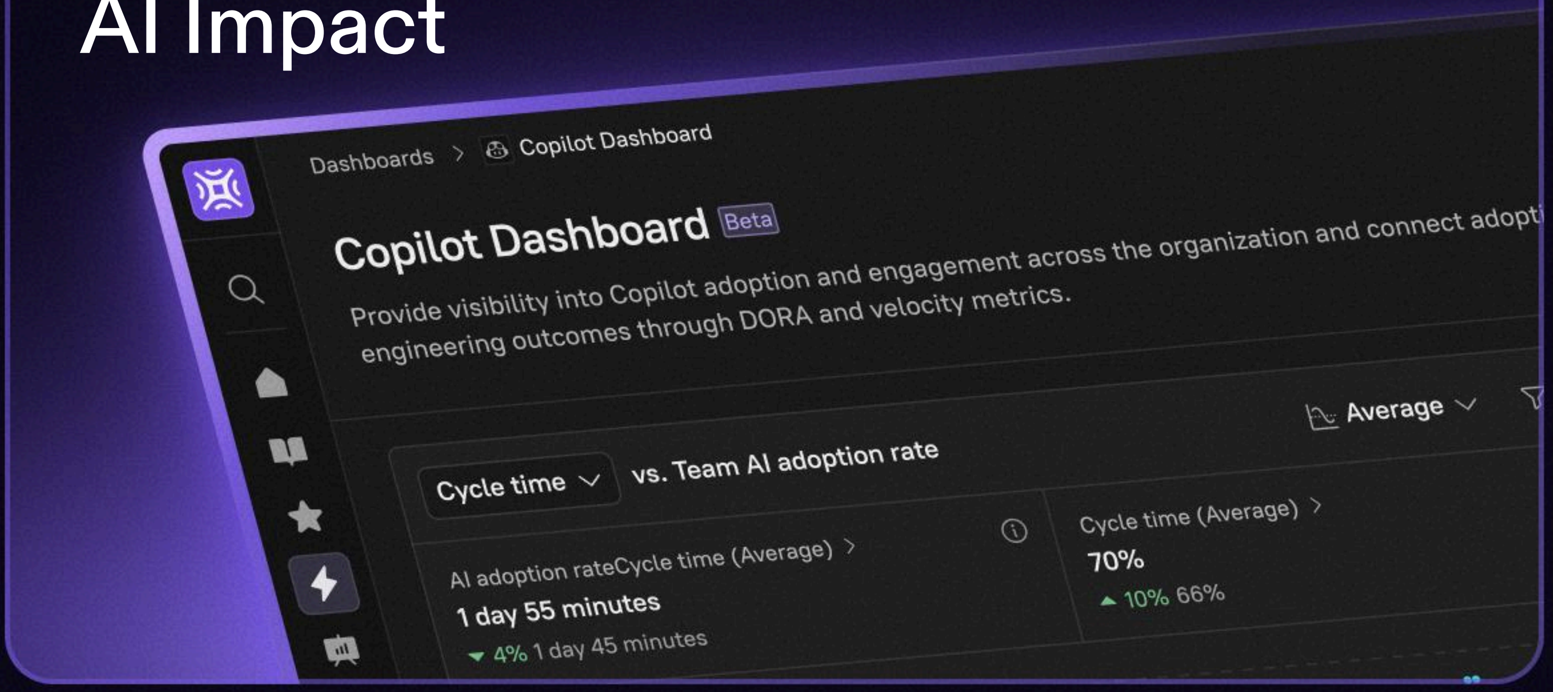
AI Readiness



Adopt AI on a stable foundation

Establish the foundations for safe AI adoption with clear ownership, strong testing, and secure CI/CD processes.

AI Impact



Provide visibility into Copilot adoption and engagement across the organization and connect adopt engineering outcomes through DORA and velocity metrics.

Prove AI's value with real metrics

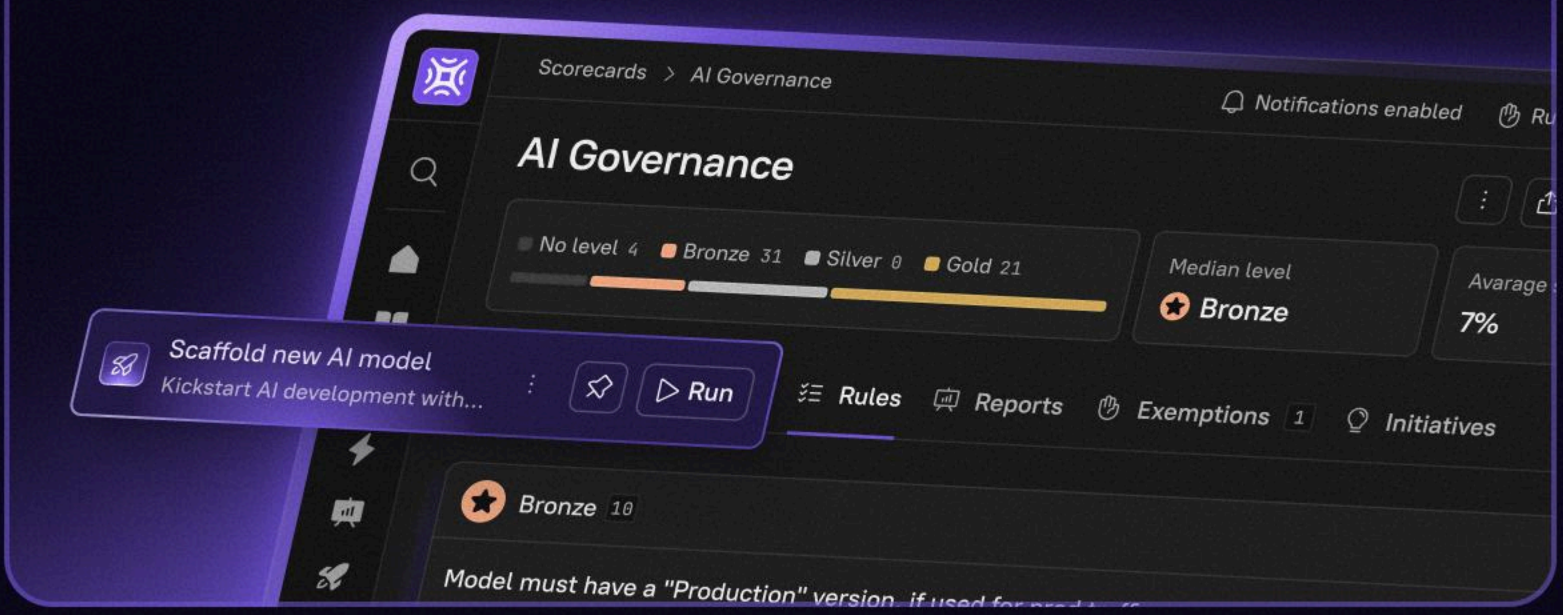
Connect AI adoption to delivery, quality, and reliability metrics to show true business value.

AI Maturity

Standardize AI adoption

Ensure every service is prepared to use AI safely and effectively with clear prompts, coding agent instructions, and monitoring rules across repos.

AI Governance



Model must have a "Production" version, if used for prod...

Turn AI from black box to business standard

Catalog and govern AI/ML models that ensure clear ownership, standardized evaluation metrics, and consistent creation processes.

Unlock Engineering Intelligence.
Get started with Cortex today.

cortex.io/demo

For more on how to track and act on these metrics, check out [Cortex's Engineering Intelligence.]()

cortex