

PoV

AI Coding Productivity

Why Writing Got Faster, But Shipping Did Not

AI writes the code. The rest of the software development life cycle did not keep up. Where the money leaks, and what closes the gap.

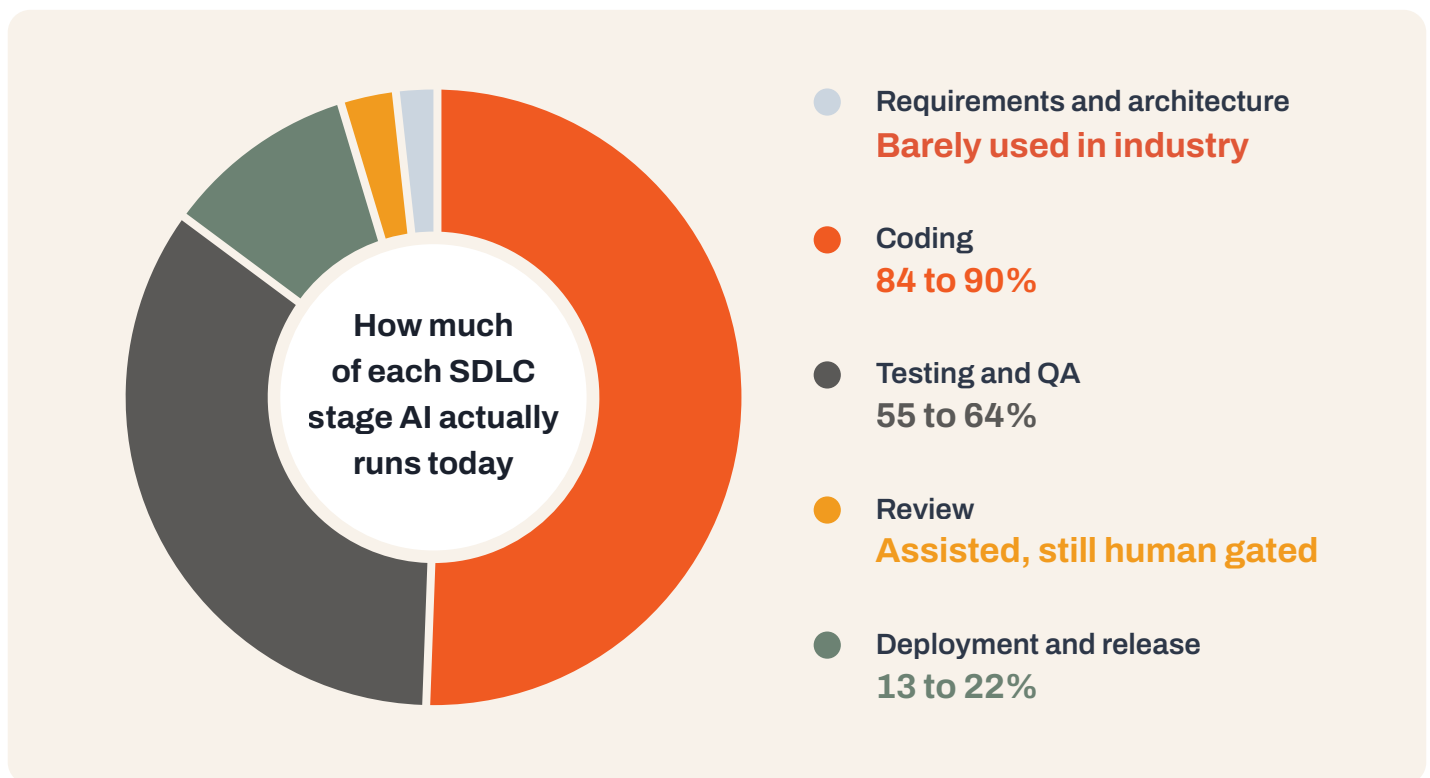
Table of Contents

- 01 Executive Summary
- 02 What AI Coding Tools Genuinely Deliver
- 03 Where the Gains Stop
- 04 5 Places The Money Leaks
- 05 What Closes It
- 06 The Way Forward - AI Engineering
- 07 Evidence base

Executive Summary

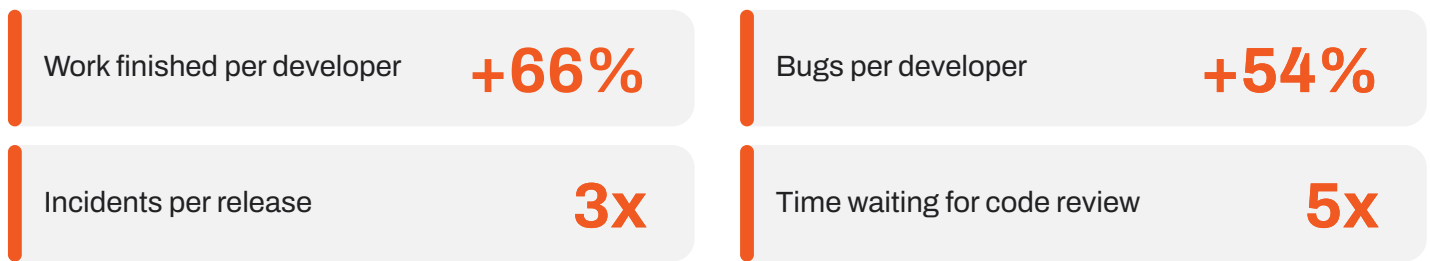
AI coding tools work. That is settled. **93% of development professionals now use AI; agentic adoption in coding has reached 84%-90%; and in committed organizations, AI writes or assists with around half the code.** At the keyboard, the problem is solved.¹

The rest of the software development life cycle did not keep pace. **Testing sits at 55% to 64% adoption. Deployment and release sit at 13% to 22%.** Requirements and architecture, the two stages that decide whether generated code fits the system at all, are barely used in industry at all.²

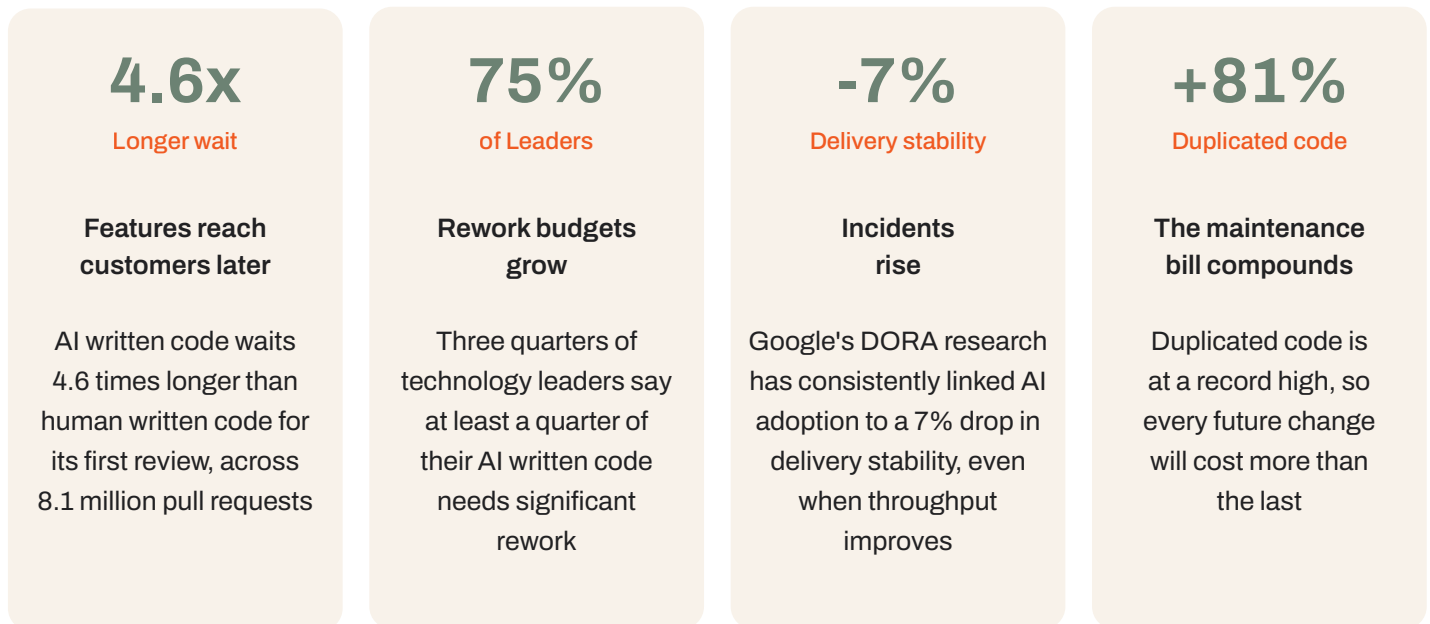


Only about 13% of teams have AI operating across the full life cycle. Most organizations sit at the first or second stage of maturity, where AI helps individuals, but the workflow, governance, and measurement around it have not changed at all.³

Coding productivity went up. SDLC productivity did not. The work moved to the two ends of the SDLC: upfront architecture and downstream review, testing, integration, governance, and adoption.

Source ⁴

What this costs a CIO

Source ⁵

McKinsey puts a number on the business result. **Only a quarter of companies report meaningful acceleration of their product development life cycle, and productivity actually fell in around 30% of companies after adopting agentic coding tools.** ⁶

The fix is not a better coding tool. **It is building the AI engineering practice that lets the whole life cycle run at the speed the coding stage does now.** This paper sets out the five stages where the money leaks, what closes each one, and what to put in place.

What AI Coding Tools Genuinely Deliver

Two problems are solved. Syntax is one. Models now produce code that compiles nearly every time, and the errors that dominated early adoption have largely disappeared. **Volume is the other.** Under high adoption, teams complete two-thirds more epics per developer than before.

The gains are largest on work that is bounded and clearly specified. A new service. A well-defined feature. A legacy translation with a clear definition of done. **Morgan Stanley aimed its tooling at one problem, documenting legacy COBOL and PL/I, and saved an estimated 280,000 developer hours.**

The ceiling is rising. **Gartner** expects agent workflows to lift team productivity by 30% to 50% by 2028, against 0% to 20% from code assistants in 2025. **McKinsey** found the top fifth of engineers already averaging 55%.

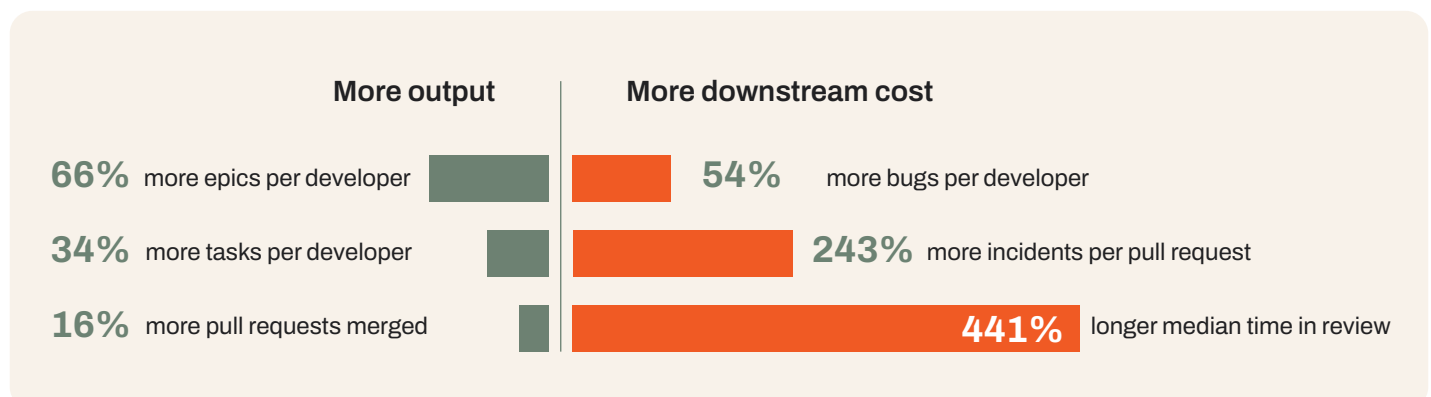
The lesson from the leaders is consistent. Narrow, well-specified work beats broad rollout. The tool performs when someone has defined the problem for it.

Where the Gains Stop

Most AI-generated code is fine. **The problem is that code can be correct in its own file and wrong for the system it lands in.** The enterprise finds out later, in review, in testing, or in production.

The tool is accurate about what you asked for. It knows nothing about the system you asked it to change.

The same industry research that shows output rising also shows where the cost went.



McKinsey reports that speed is improving faster than quality: average time savings of 11.8% against a rework reduction of only 6.2%. **Gartner** describes a new class of defect created by AI code that lacks context, and warns these defects cost far more to fix than ordinary coding errors. **Forrester** expects 75% of technology decision makers to face moderate or severe technical debt in 2026.

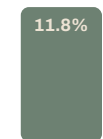
At the company level, the gain gets small. **DX** tracked more than 400 companies: with 65% of engineers using AI, median pull request throughput rose 7.8%. That is the number a CFO sees.

Speed is outrunning quality. The company gain is small.

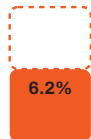
Three analyst signals, from the team to the P&L

Speed outruns quality

McKinsey, 2026



Time saved



Rework reduced

The gap comes back as rework and debt

Debt is building



Forrester
of technology leaders expect
moderate or severe technical
debt in 2026

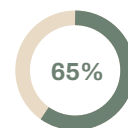
A new class of defect

Gartner, 2026

AI code that lacks system context
Cost far more to fix than a coding error

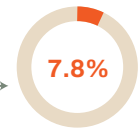
What the CFO sees

DX, 400+ companies



of engineer
use AI

becomes



median rise in pull
request throughput

That is the number a CFO sees.

5 Places The Money Leaks

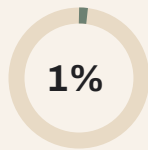
Most AI-generated code is fine. The problem is that code can be correct in its own file and wrong for the system it lands in. The enterprise finds out later, in review, in testing, or in production.

1. Requirements and architecture

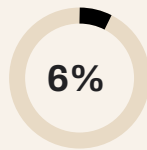
An AI coding tool only reads the small part of your codebase it has been handed. It cannot see the rest. **So it writes code that is correct on its own and wrong once it meets your other systems.** The developer sees working code. Nobody finds the real problem until testing, or until production.

The obvious fix is to give the tool more of your estate. That makes it worse. Accuracy falls as you feed in more, in every leading model tested.

How often the tool invents something false ⁷



Given the exact files it needs



Left to pick its own files



Given the whole codebase

A 2026 review of 67 studies names the **context window constraint** and the **code review bottleneck** as the two mechanisms that turn individual speed into organizational stall. Separate research found that teams with weak requirements discipline carry measurably higher technical debt, while teams with rigorous written requirements see far smaller quality trade-offs at the same velocity. ⁸

2. Review

One developer with an AI tool opens more changes in a day than a reviewer can properly read in a week. The changes are also harder to judge, because the reviewer did not write them and has no context for the decisions inside them.

Human written code



baseline wait

AI written code



4.6x longer wait

● Time waiting for a first human review, across 8.1 million pull requests ⁹

So the queue grows. **Median time waiting in review rose from 91% above baseline in 2025 to 441% in 2026**, and nearly a third more changes now reach production with nobody reviewing them. **Surveys now describe senior reviewers as the hidden integration layer of the organization.** ¹⁰

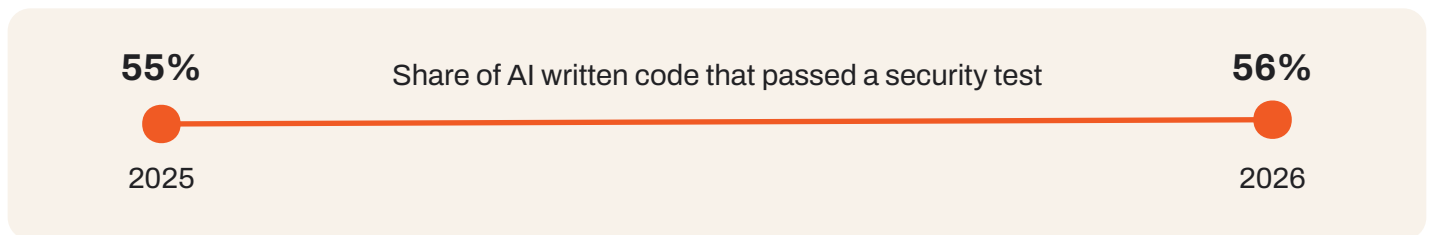
Using AI to review AI helps, and it does not close the gap. A quarter of changes are now reviewed by an agent. **Measured against problems a human reviewer found, these tools catch between 15% and 31% of them.** ¹¹

3. Testing

Testing is the second-most mature stage, with adoption ranging from 55% to 64%, and **it is still where most of the gains are lost**. The reason is that AI is good at writing tests and bad at knowing what should be tested.

Here is a change that will clear all your checks. Every line covered by a test. Every test passing. And the tests prove almost nothing **because they were written to match the existing code rather than what the business asked for**. AI-written tests achieve similar coverage to human-written ones while checking a narrower range of the most complicated changes.¹²

Non-functional requirements are the bigger gap. **Security rules, error handling, and performance limits are almost never written into the request, so they are not in the code and not in the tests**. The same security evaluation has now run twice, a year apart. In that year, models got far better at writing code that works. Security did not move.¹³



4. Integration

Integration is now the single biggest challenge cited by technology leaders working with AI coding tools, ahead of cost and accuracy. The cause sits upstream. **A tool that cannot find the function that already exists writes another one**. Payment logic that should sit in one place ends up in seven.¹⁴

Across 623 million code changes since 2023, duplicated code is up 81%, and refactoring has collapsed to under 4% of changes. Flaky integration tests then push deployment failures up, which is where a clean-looking release turns into an incident.

5. Release and adoption

Deployment and release sit at **13% to 22% agentic adoption, the lowest of any stage**. Tools now support greater autonomy than release processes allow, a trend the industry has begun to call a deployment overhang. **More code arrives at a release process built for less of it**.¹⁵

The last step is the one nobody budgets for. A release is not value until the business uses it. Organizational delivery metrics across the DORA set- lead time, deployment frequency, and change failure rate- show no measurable improvement from AI even where individual output rises sharply. **That gap between what the team produced and what the business adopted is the leak that shows up on the P&L**.¹⁶

What Closes It

The gap	The fix
01 The tool works on one file and cannot see the rest of your systems, so code that looks correct breaks something else	Write the architecture down before any code is generated. How the parts connect, what each one is allowed to do, which rules apply. Require a design review before AI starts on any significant feature. Give that document an owner and keep it current, because it goes out of date as the system changes. Where several developers or agents work on the same system, this is the only thing that stops five different versions of the same component.
02 Code arrives faster than anyone can review it, so changes reach production unreviewed	Fund review in proportion to how much code you are generating, and move the check earlier so work is reviewed as it is built rather than in a pile at the end. Use AI review as a first pass underneath standard code scanning, never instead of a person. Watch how often developers act on its comments, because a tool the team has learned to skip is costing money and catching nothing.
03 Tests pass without proving the software does what the business asked for	Judge tests on what they catch, not what they cover. Deliberately break the code and check whether the tests notice. Write security rules, error handling, and performance limits into the design so they reach the tests. On older systems, record how the system behaves today before letting any AI-assisted change touch it.
04 The same logic ends up in several places, and nothing fits together cleanly	Make existing code easy to find and shared standards easy to enforce. Search across the whole estate, document shared libraries, and automatic limits on duplication that stop the build rather than appear in a report nobody opens. Stabilize integration test suites before expanding generation capacity.
05 Software ships into a release process built for less of it, and the business does not pick it up	Bring the release pipeline up to the speed of the coding stage before adding more generation. Then treat adoption as part of delivery. Demonstrate working code and passing tests to the people who asked for the change, get sign-off on what they can see, and measure the lifecycle end to end rather than the coding stage alone.

The Way Forward - AI Engineering

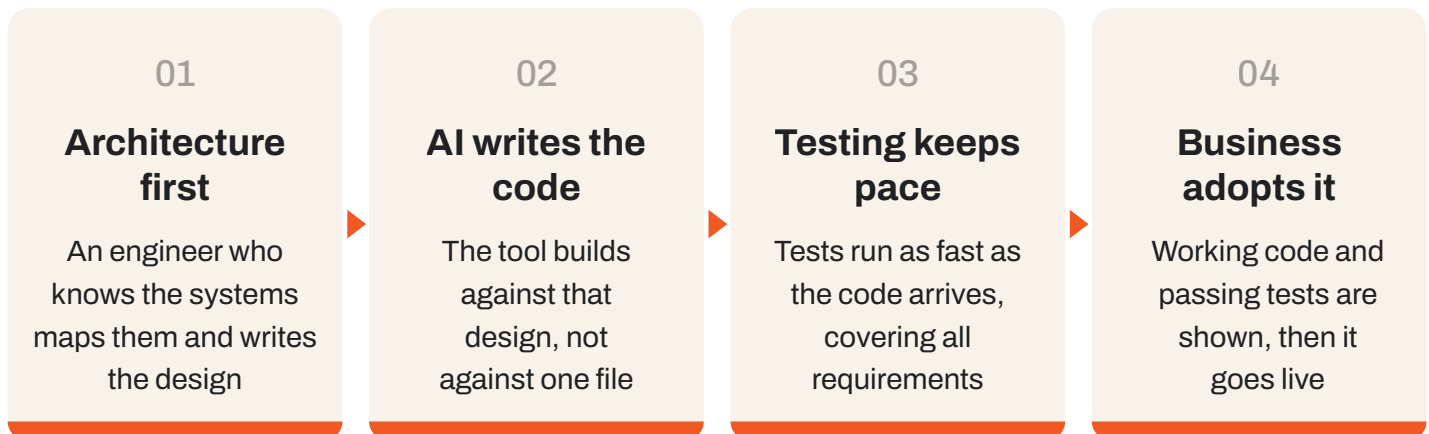
DORA's 2026 conclusion is worth stating directly, because it is the shortest summary of everything above.

AI is an amplifier. It accelerates the organizational habits you already have. Fix the review pipeline, the test estate, and the release process before adding more generation capacity.¹⁶

Gartner predicts that by 2028, teams that consistently apply an ensemble of AI-powered tools across the SDLC will achieve 25–30% **productivity gains**, up from the 10% delivered by code-generation-focused approaches in 2024.

McKinsey found that organizations that redesigned their processes before incorporating the technology were more than twice as likely to report productivity gains of more than 20%.

Agentic engineering is that redesign. It remodels the whole SDLC, from requirements to go-live to business adoption, so that agents and engineers work within a single, governed flow with clear ownership at every stage.



Each step owned end to end. The tool is one step, not the whole chain.

Architecture first

Before code is generated, someone who understands both the business requirement and the existing systems maps how the change fits and writes the design. This is a forward deployment engineer (FDE), meaning an engineer who sits close to the business, knows the domain, and owns the work from requirements through to go-live.

Without that step, the tool is guessing at everything it cannot see, which is most of your estate. This is also the least automated stage in the industry today, which makes it the clearest place to gain an advantage.

The tool writes the code

With the design in place, the coding tool does what it is genuinely good at. Fast, accurate work against a clear specification. The published velocity gains of three to five times are real in this setting. They are also the only part of the chain that is already solved.¹⁷

Testing keeps pace

If code arrives five times faster and testing runs at the old speed, the queue absorbs the gain. Testing has to be built for the new volume, and it has to cover both what the software does and how it behaves under load, under failure, and under attack.

Integration and release keep pace

The release pipeline is the least automated stage in the industry and the one most likely to swallow everything gained upstream. Stable integration suites, automated deployment, and a pipeline that can absorb the volume being generated. Without this, faster coding produces a larger queue rather than faster delivery.

Business adoption

A release is not value until the business uses it. That happens when working code and passing tests are demonstrated to the people who asked for the change, and they sign off on what they can see. Skip this, and you have shipped software nobody adopted, which is why high coding productivity so often ends in flat business results.

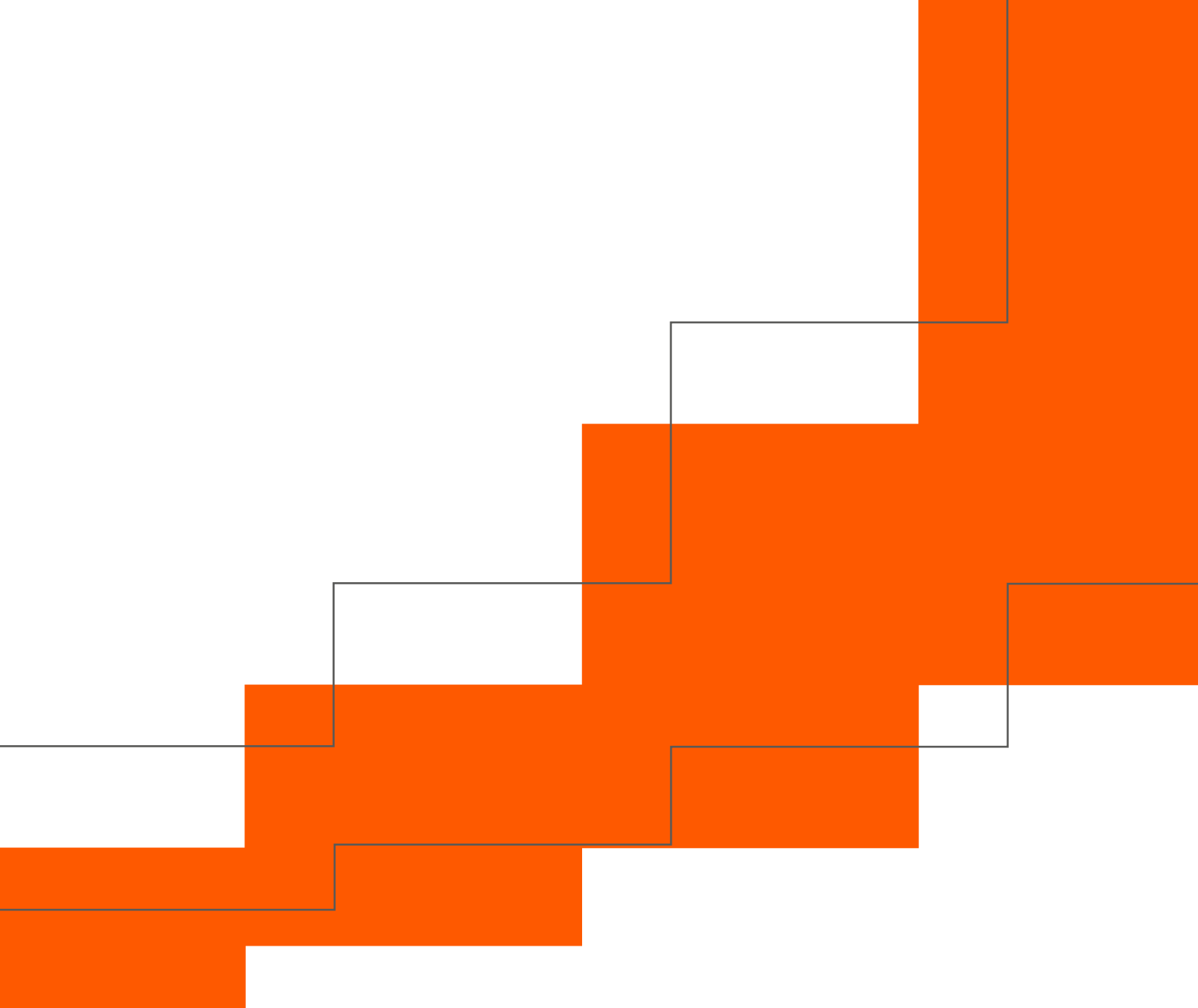
In an analyst's survey of 782 infrastructure and operations leaders, only 28% of AI use cases fully succeed and meet ROI expectations, while 20% fail outright. **The wave of agentic AI project cancellations has been attributed to three things: escalating costs without clear ROI, unclear business value, and inadequate risk controls.**

iOPEX closes this gap through agentic engineering capability built inside enterprise delivery teams. Our forward-deployed engineers own the architecture, direct AI-driven build, run requirement-led testing, and take each release through business sign-off to go live. The result is software that ships on plan, holds up in production, and gets used.

Sources

Evidence base

- 1 Vention, State of AI in Software Development, Q1 2026. Agentic AI in the SDLC, 2026 adoption data.
<https://ventionteams.com/ai/sdlc/report>
- 2 Assistance to Autonomy, systematic literature review of agentic AI across the SDLC, arXiv 2026. Agentic AI in the SDLC, 2026.
<https://arxiv.org/pdf/2605.15245>
- 3 Vention, Q1 2026. <https://ventionteams.com/ai/sdlc/report>
- 4 Faros AI, The Acceleration Whiplash, 2026, telemetry across 22,000 developers and 4,000 teams.
<https://www.faros.ai/blog/ai-code-quality-senior-engineer-review-burden>
- 5 LinearB, 2026. New Relic, 2026 State of AI Coding. DORA, Google Cloud. GitClear, The Maintainability Gap, 2026.
<https://linearb.io/resources> <https://www.faros.ai/blog/ai-code-quality-mirage> <https://dora.dev/insights/>
- 6 McKinsey, Technology Trends Outlook 2026.
<https://www.mckinsey.com/capabilities/mckinsey-digital/our-insights/the-top-trends-in-tech>
- 7 Chroma context rot evaluation, 18 frontier models. <https://research.trychroma.com/context-rot>
- 8 The Productivity Reliability Paradox, arXiv 2026. <https://arxiv.org/html/2605.01160v1> Spec Driven Agentic Development, arXiv 2026. <https://arxiv.org/pdf/2608.20341>
- 9 LinearB analysis of 8.1 million pull requests across 4,800 teams, 2026. <https://linearb.io/resources>
- 10 Faros AI, 2026. <https://www.faros.ai/blog/ai-code-quality-senior-engineer-review-burden>
- 11 SWE-PRBench, March 2026. <https://arxiv.org/abs/2607.03316>
- 12 Testing with AI Agents, arXiv 2026. <https://arxiv.org/abs/2603.13724> Harman et al., Mutation Guided Test Generation at Meta, ACM FSE. <https://arxiv.org/abs/2501.12862>
- 13 Veracode, GenAI Code Security Report, 2025 and 2026.
<https://www.veracode.com/blog/2026-genai-code-security-report-ai-risk/> Verizon Data Breach Investigations Report, 2026.
<https://www.verizon.com/business/resources/reports/dbir/>
- 14 57% of technology leaders name AI integration as the biggest challenge.
<https://blog.exceeds.ai/dora-metrics-engineering-effectiveness/> GitClear and GitKraken, 623 million code changes, 2026.
https://www.gitclear.com/ai_assistant_code_quality_2025_research
- 15 Agentic AI in the SDLC, 2026.
<https://keyholesoftware.com/agentic-ai-in-the-software-development-lifecycle-2026-adoption-and-impact-data/> SDLC AI Radar 2026. <https://www.ltm.com/insights/reports/sdlc-ai-radar-2026>
- 16 DORA, The ROI of AI-assisted Software Development, Google Cloud, 2026. <https://dora.dev/insights/>
- 17 Spec Driven Agentic Development, arXiv 2026. <https://arxiv.org/pdf/2608.20341>



About iOPEX

iOPEX Technologies is a new-generation agentic AI and automation-led enterprise transformation partner headquartered in San Jose, California. At the intersection of enterprise operations, agentic AI, and intelligent automation, we deliver Intelligence as a Service. Over 70 global brands trust iOPEX as a strategic partner to turn AI into results that scale. Learn more at www.iopeex.com.

Contact us ▶

