DX

# AI and engineering velocity: A longitudinal analysis

# AI and engineering velocity: A longitudinal analysis

Why measurable engineering velocity gains from AI are lower than expected and what leaders can do about it.

Many leaders feel their organizations are falling behind in the race to unlock AI-driven engineering velocity. Vendor marketing and social media set expectations at 3x or even 10x improvements. When leaders see more modest results, they assume something is wrong.

Their concern is understandable: new AI models and coding tools launch frequently, often with sensationalist claims. Leaders need a clear, objective picture of how AI is actually impacting engineering velocity across the industry—so they can calibrate expectations, make the right investments, and stop benchmarking against hype.

To provide that picture, DX analyzed engineering velocity from November 2024 to February 2026 across a sample from 400+ companies where AI adoption rose sharply. We found a 10–15% increase in PR throughput—a real gain, but well below what most leaders expect.

To understand why, we conducted qualitative interviews with developers from these companies. This paper shares what we found, what's limiting AI gains, and what leaders can do about it.

## What the data shows

We selected a random sample from 400+ companies meeting these criteria:

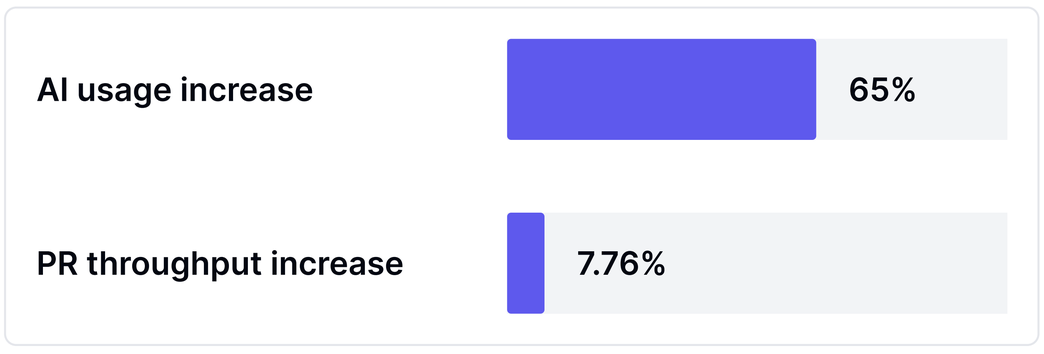
- 75% average adoption of AI coding tools based on monthly active usage
- 100+ engineers
- No known individual targeting of PR throughput (to filter out gamification effects)
- No significant M&A, regulatory changes, or IPO during the study period

The sample spans technology, financial services, retail, and healthcare companies ranging from approximately 150 to 10,000 engineers. We analyzed engineering velocity by measuring the change in merged PR throughput per engineer per month. We also validated findings against TrueThroughput, a DX metric that uses AI to weight each PR by relative size and complexity. Both signals showed consistent trends; we chose PR throughput as the primary metric for its accessibility and because it reflects how most organizations are currently tracking AI's impact.

During the study period, AI tool usage increased by an average of 65%. Median PR throughput increased by 7.76% (Figure 1).

Figure 1. AI usage increase versus PR throughput gains

Median values across companies with 80%+ AI coding tool adoption over a 12-month period




The mean PR throughput gain was 13.1%, higher than the median of 7.76%. This difference is driven by a small number of high-performing companies. Figure 2 shows the spread across companies.

Figure 2. Distribution of PR throughput gains with AI

Percentile breakdown of year-over-year change in merged PRs per engineer per month


The majority of companies saw a 2–17% increase in PR throughput as AI usage increased. Even at the 90th percentile, where gains approached 44%, results remain far below the order-of-magnitude improvements often discussed in executive circles. We have not yet validated whether gains at the high end of the distribution reflect durable, real throughput improvements or are driven by confounding factors. We plan to investigate this further in upcoming research.

For most organizations, today's AI coding tools are delivering a 5–15% throughput gain. Leaders whose numbers fall in this range are not behind. They are in line with the industry.

# The factors keeping AI gains below expectations

If AI adoption has increased by 65%, why has throughput moved less than 8% at the median?

We interviewed developers across our sample to understand what's limiting gains. Five themes came up consistently.

**1. Coding isn't the main bottleneck.** AI accelerates code generation, but coding is a relatively small share of how engineers spend their time. Microsoft's research puts coding  at approximately 14% of a developer's typical day. Even cutting that time in half wouldn't meaningfully move the overall needle.

As one developer put it: "The easy tasks are a little easier. The tedious tasks are a little less annoying. A four-day task might take three. But that doesn't mean I'm shipping 3x more PRs." Another senior engineer: "Many senior devs say: writing code is the easy part. The bottleneck is the human side: alignment, planning, scoping, code reviews."

**2. Speeding up one part of the SDLC creates bottlenecks in others.** AI has accelerated code generation, but many downstream processes haven’t scaled with it—code review and integration remain largely unassisted. Several developers described a dynamic where the time saved writing code is consumed by the extra scrutiny that AI-generated output requires.

Table 1. Factors limiting measurable productivity gains from AI

| Themes from qualitative interviews with developers | | |
|---|---|---|
| **Bottleneck** | **Description** | **Representative quote** |
| **1. Coding is not the primary bottleneck** | Coding represents ~14% of a developer's time. Accelerating it yields single-digit improvements to overall throughput. | *"A four-day task might take three. But that doesn't mean I'm shipping 3x more PRs."* |
| **2. Automation creates new bottlenecks** | Time saved generating code is offset by increased review, validation, and remediation burden. | *"AI can significantly speed up initial engineering time, but often that saved time is spent on extended reviews, fact checking, or issue remediation, resulting in net-zero productivity gain."* |
| **3. Social friction inhibiting adoption** | Pro/anti-AI polarization and unclear norms inhibit teams from developing shared AI workflows. | *"Being an isolated solo-adopter does not allow you to materialize the gains in a meaningful way. Software development is a team sport."* |
| **4. Compounding skill and tooling gaps** | Immature tools steepen the learning curve; developers early on the curve extract less value from the tools. | *"I have 1,000+ hours of purely agentic coding experience and I have so much to learn. Many beginners may have just a few hours or tens of hours."* |
| **5. AI tools lack context** | Institutional knowledge is not accessible to AI tooling. | *"An AI assistant can't reason over a Slack thread from an archived channel or the mental model of the engineer who built it."* |

One developer said, "AI can significantly speed up initial engineering time, but often that saved time is spent on extended reviews, fact checking, or issue remediation, resulting in net-zero productivity gain." Another: "People do not have all parts of the SDLC AI-enabled. Wherever it does not exist becomes a bottleneck—planning, code review, verification."

**Three additional themes surfaced:**

- **Skill and tooling gaps are limiting gains.** Using AI effectively is its own discipline, and most developers are still early in developing it. At the same time, the tools themselves aren't always applied to the use cases where they're the best fit. These two factors reinforce each other: ill-suited tooling steepens the learning curve, and developers early on the curve get less out of the tools.
- **AI tools lack critical context.** AI performs well on self-contained, well-documented problems. Most real engineering work is neither. The context typically lives in people's heads or is distributed across heterogeneous systems of record. Until that knowledge is made explicit and accessible, AI will hit a ceiling on the kinds of problems it can reliably help with.
- **Social friction slows adoption.** Polarization between pro- and anti-AI engineers, ambiguity about when and how AI should be used, and the absence of peer champions all slow the rate at which teams develop effective workflows with these tools.

While these five factors help explain the gap between expectations for AI and reality, they don't fully account for it, particularly the question of where developers are reinvesting time saved by AI. Our data shows developers self-reporting meaningful time savings, but those savings are not showing up proportionally in output. Where that time is going remains an open question that requires further research.

## Unlocking further gains: Three steps for AI investment

To help leaders unlock more from their investments in AI, we recommend three steps: (1) strengthen foundational readiness before scaling AI, (2) identify where AI can deliver additional value, and (3) measure to ensure gains are not coming with unwanted tradeoffs.

### 1. Build foundational readiness

AI tools are amplifiers. They take what developers are already doing and scale it, for better or worse. But what determines the output isn't the tool or the environment alone; it's how developers use AI within that environment. A well-documented, well-structured codebase with low friction makes it easier for developers to use AI effectively. Poor documentation and high friction make it harder, not because AI mechanically reproduces bad code, but because developers working in those conditions have less to steer with.

This means organizations should invest in ensuring the codebases and workflows where AI is being introduced are well-maintained and well-documented. The goal is to create the conditions where developers can get the most out of these tools. Table 2 provides examples of areas that are likely to impact AI tool effectiveness; we recommend organizations begin assessing them at the service and team level before scaling AI use. We're continuing to research what foundational readiness looks like in practice and will share updated guidance as the data develops.

Table 2. Foundational readiness areas for AI-assisted engineering

| Domain | What to assess | Why it matters for AI |
|---|---|---|
| **Validation maturity** | ✓ Code coverage meets threshold<br>✓ Automated linting and formatting standards are enforced<br>✓ Repository has had commits in last 90 days<br>✓ Module dependencies are up to date<br>✓ Code complexity is optimized for agent readability | AI tools perform best on well-maintained codebases with consistent patterns. Services with declining coverage, stale repositories, or poor documentation produce lower-quality AI output and higher remediation burden. |
| **Documentation & context** | ✓ AGENTS.md (or equivalent context) is present<br>✓ README is present<br>✓ Architecture decision records exist<br>✓ API schema documentation exists<br>✓ Documentation updated within last 90 days<br>✓ Onboarding guide exists | As our interview data showed, AI tools hit a ceiling when the context they need lives in people's heads. Documentation does double duty: it improves developer experience directly, and it provides the context layer AI tools need to produce relevant output. For organizations adopting AI agents, this extends to machine-readable context files that agents can consume programmatically. |
| **CI/CD feedback loops** | ✓ Optimal CI/CD pipeline<br>✓ Build time within threshold<br>✓ Test flakiness rate below threshold | AI can generate code quickly, but if builds are slow, tests are flaky, or deployment pipelines are unreliable, the time saved writing code is consumed downstream. Fast, reliable feedback loops are what allow AI-driven speed to translate into throughput, and they are the primary guardrail against AI-generated defects reaching production. |
| **Standards** | ✓ No critical/high vulnerabilities beyond SLA<br>✓ Security scorecard checks passing<br>✓ Dependency vulnerabilities remediated within policy<br>✓ Compliance standards met for service tier | AI-generated code introduces a larger surface area of output that must meet security and compliance standards. Without automated checks and enforcement, the increased velocity from AI tools can accelerate the introduction of vulnerabilities faster than teams can catch them. |

## 2. Identify where to apply AI and what should remain human-led

Once foundational readiness is established, leaders should evaluate where AI can deliver value beyond code generation. Coding represents only ~16% of a developer's time; the highest-leverage opportunities are likely elsewhere.

The starting point is understanding where engineers are experiencing the most friction. Google's developer productivity research team describes this approach: they use periodic developer experience surveys to identify the top hindrances to productivity, then direct AI investment toward those specific pain points. We recommend this data-driven approach.

This assessment must be ongoing, not one-time. Accelerating one part of the SDLC can create bottlenecks in others. Organizations need to assess flow regularly to detect when AI-driven speed in one area is producing new constraints downstream.

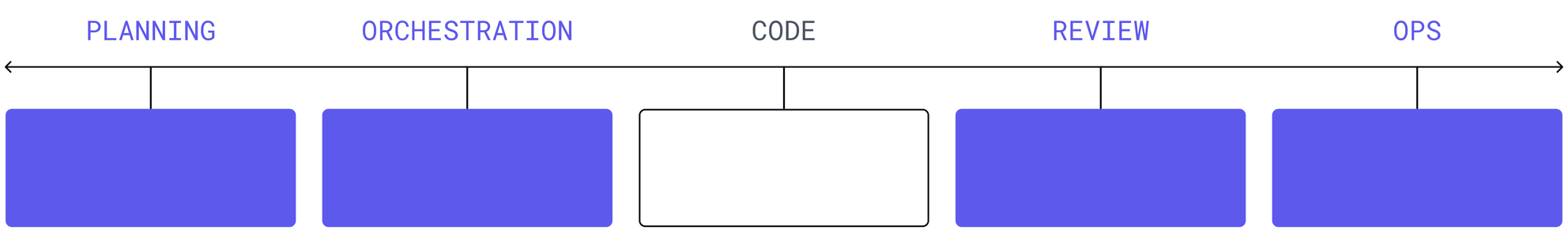
With this data in hand, leaders can evaluate potential AI use cases across three categories:

- **Developer friction.** Where are developers experiencing the most friction, and where do they most want investment? Capture this through developer experience surveys, system metrics, and telemetry data.
- **AI readiness.** How well-suited is the infrastructure to current AI capabilities? Gartner's research on AI-native software engineering recommends assessing suitability across three factors: whether context is readily accessible, whether the task is tightly scoped with clear outcomes, and the business criticality and risk threshold.
- **Current AI penetration.** An area that scores high on suitability but has already seen heavy AI adoption—code generation being the clearest example—is likely to yield diminishing returns compared to an equally suitable area with little AI penetration today.

### What should remain human-led?

Not every stage of the development lifecycle can currently be automated. AI can draft a test suite or generate boilerplate documentation, but scoping a quarter's roadmap or evaluating an architectural trade-off requires judgment today's models cannot reliably provide. Leaders need to be intentional about what stays human-led.

Figure 3. Untapped opportunities lie across the broader SDLC



## 3. Measure gains and trade-offs

Increased velocity from AI comes with risks that, left unmonitored, can erode the gains. Leaders should track not just whether throughput is improving, but whether that improvement is coming at a cost.

We recommend using the AI Measurement Framework (Table 4) to track AI's impact across three dimensions: utilization, impact, and cost. This multi-dimensional approach is essential because improvements in one dimension can come at the expense of another.

As agents take on a larger share of engineering work, we expect our measurements to evolve alongside them. One emerging metric is **agent hourly rate:** the human-equivalent hours an agent delivers, divided by its cost. When that effective rate is low relative to human hourly rate, it signals a high-ROI use case worth scaling. More broadly, we're beginning to explore measurement approaches that treat agents less like tools and more like teammates, assessing factors like how well agents can parse requirements, navigate a codebase, and respond to human steering.

Just as developer experience measurement improved engineering effectiveness by surfacing friction developers face, similar approaches applied to agents can help organizations identify what's limiting agent effectiveness and where to invest. We expect the AI Measurement Framework to continue evolving as these practices mature.

Beyond these core metrics, leaders should watch for three risks that our research has shown to accompany AI-driven velocity gains:

**Defective code.** AI-generated code can introduce defects that are difficult to catch, particularly in complex production systems. Amazon's experience in early 2026 is instructive: AI-generated code contributed to outages that resulted in approximately 120,000 lost orders in one incident and a 99% drop in North American orders in another. Amazon responded with a 90-day safety reset, mandatory two-person code review, and audits across 335 Tier-1 systems. An internal review noted: "GenAI's usage in control plane operations will accelerate exposure of sharp edges and places where guardrails do not exist." If Amazon—one of the world's most sophisticated engineering organizations—is experiencing this, every organization should be paying attention.

Table 3: DX AI Measurement Framework

* Metrics for autonomous AI agents

| **Utilization**<br>How much are developers adopting and utilizing AI tools? | **Impact**<br>How is AI impacting engineering productivity? | **Cost**<br>Is our AI spend and return on investment optimal? |
|---|---|---|
| • AI tool usage (DAUs/WAUs)<br>• Percentage of PRs that are AI-assisted<br>• Percentage of committed code that is AI-generated<br>• Tasks assigned to agents * | • AI-driven time savings (dev hours/week)<br>• Developer satisfaction<br>• DX Core 4 metrics, including:<br>  • PR throughput<br>  • Perceived rate of delivery<br>  • Developer Experience Index (DXI)<br>  • Code maintainability<br>  • Change confidence<br>  • Change fail percentage<br>• Human-equivalent hours (HEH) of work completed by agents * | • AI spend (both total and per developer)<br>• Net time gain per developer (time savings - AI spend)<br>• Agent hourly rate (HEH / AI spend) * |

**Cognitive debt.** As AI accelerates code production, teams risk losing shared understanding of their own systems. Dr. Margaret-Anne Storey, co-author of the SPACE and DevEx frameworks, has describe this as cognitive debt: the erosion of the collective mental model of what a system does, how it was designed, and how it can be safely changed. Unlike technical debt, which lives in the code, cognitive debt lives in people. It could manifest as loss of confidence when making changes, heavier review burden, debugging friction, slower onboarding, and increased stress.

Whether cognitive debt proves to be a material risk is still an open question. Some argue that developers can use AI itself to quickly regain understanding of systems when needed, reducing the cost of lost familiarity. Others maintain that deep human understanding remains essential: that if you need to call the mechanic, they need mastery over the machine. Our interview data surfaced early signals worth watching, but we don't yet have a definitive answer.

**False velocity.** There is a meaningful difference between producing more code and delivering more value. More PRs do not necessarily mean higher business velocity. Developers and teams may also over-report the perceived benefits of AI tools—a dynamic documented in METR's research. Leaders should distinguish between activity and impact, and ensure that throughput increases correspond to real progress on business outcomes.

## Where to start

The gains we see today—a median of 7.76%, with most organizations in the 5–15% range — are meaningful but modest. An organization with 500 engineers seeing a 10% improvement is getting the equivalent output of 50 additional engineers without the headcount cost. That is a real return. The mistake is measuring it against an expectation of 2–3x.

These gains are also not the ceiling. Closing the gap between current results and the potential of AI will require looking beyond code generation.

We recommend three steps. First, assess your foundational readiness by service and team, and address the gaps that limit what AI can deliver. Second, use the AI use case evaluation to identify where in your SDLC the highest-leverage opportunities exist, and where human judgment should remain central. Third, measure progress to track gains and potential tradeoffs.

The organizations that capture the next wave of AI-driven productivity gains will be the ones that invest in the full breadth of their development lifecycle — and in the human and contextual conditions that allow AI to be effective.

DX ANNUAL
presented by DX

Rewatch the discussion of these findings here.

# About the author

Abi Noda is co-founder and CEO of DX where he leads the company's strategic direction and R&D efforts. Justin Reock is Deputy CTO at DX, focused on transforming businesses through developer productivity theory, AI research and strategy, developer experience optimization, public speaking, training, and enablement.

## Engineering intelligence

Learn more at getdx.com