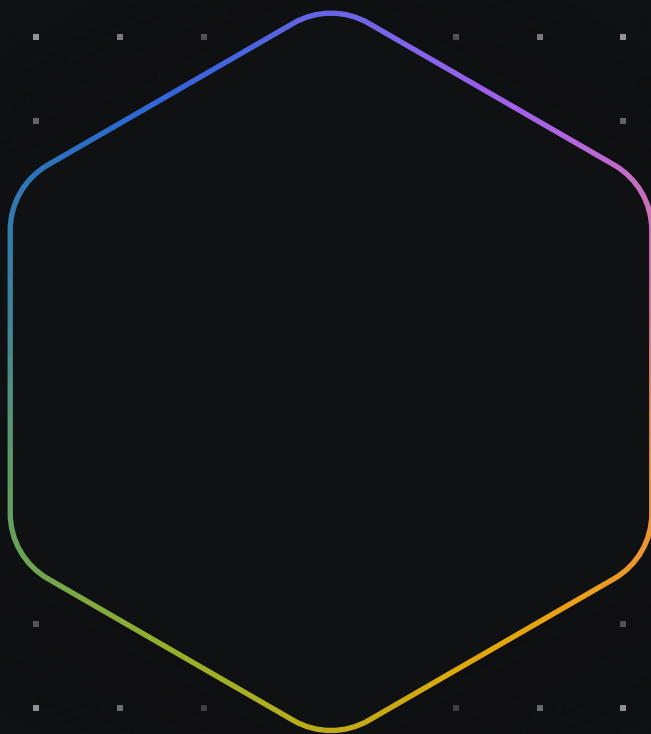


The Agentic Pivot



Insights from 1,000+ engineers
and leaders on the shift to agentic
software development.

Overview

AI-assisted software development is now the norm. Almost all engineering leaders (94%) say they use AI somewhere in their engineering organization. A little over a third (36%) already run agentic workflows that operate with limited human intervention.

But despite the rapid speed of AI adoption, development bottlenecks still remain, and they're shifting. Across a 2026 survey of more than 1,100 engineers and engineering leaders, one pattern shows up over and over: AI has compressed the implementation phase of the software development lifecycle—code creation—while not yet increasing velocity for the phases left-of and right-of code, creating new bottlenecks in the SDLC.

Nowhere is this more visible than in code review: 74% of teams say AI accelerate code generation, yet nearly 80% of teams still review AI-generated code through the exact same manual PR process from a decade ago, a review process designed for a different volume, a different velocity, and a fundamentally different kind of author.

To effectively address these new bottlenecks in the next 12 months, engineering organizations need to turn their focus from individual AI adoption towards formalizing AI-native software delivery across the organization—building an engineering system of work that provides the shared context, governance, and judgment that human-and-AI teams now depend on.

In our survey, engineering leaders already sense this shift: 88% say they need a governed system of record; only 19% have built one.

Here are the five additional findings that define the pivot:

- 1 Adoption is universal but experimental. Formalization is rare.
- 2 Context and governance remain the biggest hurdles to AI formalization.
- 3 The definition of the engineering role is becoming broader.
- 4 Engineering leaders want AI's time-savings reinvested in better-quality software, not headcount reduction.
- 5 The biggest SDLC opportunities are left-of and right-of code: planning, testing, reviewing, and shipping.



This report is based on responses from more than 1,100 global software professionals across individual contributors and engineering leaders (including CEOs, CTOs, VPs, Directors, and others). See the appendix for detailed methodology.

Adoption is universal but experimental. Formalization is rare.

EVERYONE IS USING AI INDIVIDUALLY. ALMOST NO ONE HAS FIGURED OUT HOW TO USE IT TOGETHER.

In our research, 94% of engineering leaders say their teams use AI, but usage is extremely broad with varying degrees of autonomy. It counts anyone whose engineering organization uses AI in any form, from IDE autocomplete to browser-based chatbots to production-integrated agents. The more useful question is how leaders describe that use:

88%

report AI supporting individual tasks like coding, debugging, and documentation—the "AI pair programmer" mode

54%

report AI supporting team collaboration and knowledge sharing

36%

report agentic workflows, AI that runs automatically with limited human intervention

The 94% includes all three modes. The 36% agentic figure is the more meaningful marker of how far beyond autocomplete the average engineering organization has actually moved.

Formalization tells a different story

Adoption at some level is nearly universal. Adoption as an official standard is not.

How AI is integrated into engineering workflows	Improvement
Experimental and limited (informal, few areas of the SDLC)	27%
Experimental but extensive (informal, many/most areas of the SDLC)	39%
Formal but limited (official standard, few areas of the SDLC)	28%
Formal and extensive (official standard, many/most areas of the SDLC)	6%



The difference between an experiment and a standard isn't measured in usage, but in downstream effects when the tool fails, when the team turns over, when the model changes, when auditors ask, or when a decision made by an agent six months ago needs to be understood by a human today. Most engineering organizations have not yet had to answer those questions at scale.

The stakes are rising

Agentic development is no longer a distant capability. More than a third of engineering organizations already run agentic workflows—and individuals within those organizations are adopting agentic patterns on their own, with or without shared standards to govern them. The window to move from individual adoption to a shared engineering system of work is closing faster than the industry conversation suggests.



At Atlassian, 52% of vulnerabilities are now automatically resolved through an agentic workflow with minimal engineer intervention.

Context and governance remain the biggest hurdles to AI formalization.

CODE GENERATION IS UP. TRUST IN THAT CODE IS NOT. AND THERE ARE TWO SPECIFIC REASONS WHY.

AI code generation is accelerating across every dimension the survey measures. Trust in that code—the willingness to let it move through the SDLC without a human check—has barely moved. The gap between the two is not a mystery. It is the direct result of two specific hurdles that most engineering organizations have not yet addressed at scale:



The governance around AI-generated work is essentially unchanged, and most organizations in the survey haven’t yet built the connected context layer AI needs to be trusted at scale.

The trust ceiling is real

How much does your organization trust AI-generated code?	Leaders	ICs
Zero trust (mandatory line-by-line review)	25%	32%
Guarded trust (non-critical code with minimal review; core logic reviewed)	62%	60%
High trust (AI output trusted by default)	12%	8%
Full autonomy (AI ships to production, no manual gate)	1%	<1%

Full autonomy is essentially nonexistent, and the dominant posture is guarded trust: AI can generate the code, but humans must still review it.

That posture is understandable but unsustainable. When 74% of leaders report accelerated code generation without also accelerating the review process, the throughput gain doesn't compound; it queues.



In the Q2 2026 State of AI Impact in Engineering Report, Atlassian DX research compared code maintainability (the ease with which code can be understood by developers) and change confidence (whether developers feel confident making changes to that code without accidentally breaking things).

Between Q2 and Q1 of 2026, code maintainability went up (3.8%) while change confidence went down (-6.1%). This means while AI is making it easier for developers to understand what's in front of them, the data suggests that they have less trust in the code they are actually pushing into production.

Hurdle #1 – The governance gap

Governance approach towards AI-generated code	Leaders	ICs
Traditional peer review (unmodified PR process)	78%	79%
Automated guardrails (AI scanners, linters, sandboxes)	35%	20%
Shift-left governance (prompt libraries, spec-first, AI templates)	18%	13%
Decentralized / ad hoc (no formal guardrails)	17%	22%

Nearly 8 in 10 teams review AI-generated code the same way they've always reviewed human-generated code. Automated guardrails built to catch AI-specific failure modes are in place at only about a third of leader orgs, and shift-left governance is rarer still.

About 1 and 5 teams have no formal guardrails for AI-generated code at all, leaving the quality bar entirely to the individual developer's discretion. That absence of guardrails puts a corresponding weight on observability instrumentation—the ability to catch AI-generated issues in production, when the upstream systems that would have caught them earlier don't yet exist.

Hurdle #2 – The context gap

The second reason why trust in AI-generated code isn't scaling is a lack of context.

Code review is the clearest example: Acceptance criteria live in tickets that agents don't read; engineering standards live in wiki pages that agents don't access; prior architectural decisions live in Slack threads and human heads.

When AI writes a function, it has the local code context, but not the organizational context that a senior engineer brings automatically: what shipped last quarter, what broke in incident #4237, what the security team said to never do again.

An AI reviewer without that context can only check syntax and code quality.



Atlassian Teamwork Graph is a live, intelligent map of your organization's people, work, and knowledge. It gives coding agents the precise context they need to deliver more accurate results with fewer tokens.

In Atlassian internal testing, AI agents that leverage Teamwork Graph context achieved a 44% improvement in answer quality using 48% fewer tokens.

The consequence: the reasoning gap

Governance and context missing at scale produce a specific downstream cost: the reasoning behind AI-assisted decisions gets lost. When AI is asked to explain a decision, it isn't retrieving a record of its process or previous routine, but a plausible account of what its process might have been when initially prompted. Humans must lead reverse-engineering efforts to explore and audit an agent's reasoning, a process referred to as reasoning reconstruction.

92% of ICs say AI-generated code needs at least some human review before it merges to production.

Confidence in reconstructing AI-assisted reasoning after 6 months	Leaders	ICs
Very confident	25%	15%
Somewhat confident	56%	63%
Not very confident	17%	20%
Not confident at all	2%	2%

Leaders are more confident than ICs on reasoning reconstruction, which likely reflects distance from the code more than better instrumentation. The people closest to the AI output are the least confident it can be explained later.

The distribution is telling. Only 2% of respondents in either group are not confident at all—the problem isn't fear, it's uncertainty. Most engineers can probably reconstruct the reasoning, but they wouldn't stake anything on it.

What engineering organizations must work towards is a system supporting full traceability and explainability. Until then, they'll continue to rely on human audit for real accountability.



The definition of the engineering role is becoming broader.

THE ERA OF ROLE-SPECIALIZED, PHASE-BASED SOFTWARE DEVELOPMENT IS DISSOLVING.

For decades, the software delivery model was a sequence of distinct roles owning distinct phases: PMs wrote requirements, engineers implemented, DevOps deployed, ops monitored, and QA verified. AI is dissolving the walls between those phases faster than the survey data captures. Product designers are shipping working prototypes. Engineers are doing more product thinking. PMs are writing acceptance criteria that agents consume directly. The circles that used to be sequential and specialized are now overlapping.

The split that isn't a contradiction

Ask engineering leaders how AI has changed the scope of a typical engineer's role, and the answers split three ways:

92% of ICs say AI-generated code needs at least some human review before it merges to production.

44% of leaders (and roughly the same share of ICs) say scope is unchanged

33% say scope is now broader and more generalist

23% say scope is now narrower and more specialized

That's not a signal that roles haven't changed. It's a signal that the shape of the change is being missed by the way we usually talk about it. If you ask "is your job the same?"—often yes. If you ask "has the surface area of your job expanded?"—often yes as well. The role hasn't changed titles, but it has changed weight.



The skills leaders say are more important now

When we asked leaders which skills matter more now because of AI, the responses were remarkable in how flat they were:

What skills matter more because of AI?	Leaders	ICs
Effective and secure use of AI and automation in the development workflow	69%	45%
Deep technical excellence (core stack, system design, secure coding, testing)	54%	63%
Product thinking and customer-centric decision-making	52%	52%
Ownership mindset and continuous improvement	51%	47%
Cross-team collaboration and clear technical communication	50%	42%

AI fluency leads for leaders, but every other category is within twenty points. Leaders are not saying learn AI at the expense of everything else. They are saying every existing engineering skill still matters, and AI fluency is now added to the list, not substituted for it. Engineers themselves rank deep technical excellence higher than AI fluency: leaders are saying get good at AI; engineers are saying make sure the AI is good. Both are right. Both are part of the same shift.

The judgment premium

As implementation gets cheaper, the judgment around it—what to build, what to trust, what to keep, what to remember—gets more expensive. It becomes the scarce resource.

The judgment premium

As implementation gets cheaper, the judgment around it—what to build, what to trust, what to keep, what to remember—gets more expensive. It becomes the scarce resource.

Three data points make this concrete:

80% of leaders still rely more on human judgment than AI for security and compliance

68% still rely more on human judgment for architectural decisions and system design

62% still rely more on human judgment for product scope and requirements

These are not areas where leaders are refusing to use AI. They are areas where the decision quality matters more than the execution speed. Direction becomes the scarce input, not implementation. And direction is precisely what the expanded role is being asked to provide.

That upstream shift has a specific shape in the data. The phases AI has meaningfully compressed—implementation, debugging, documentation—are precisely the phases where the judgment premium is lowest. The phases where humans still hold the judgment—problem definition, architecture, security—are precisely the phases the "dissolving" of role boundaries is exposing to the widest reshape. The dissolution of phase boundaries isn't uniform. It's happening fastest where implementation ran the phase, and slowest where judgment ran it.

Junior engineers are ramping faster

70% of leaders and 67% of ICs perceive that AI has accelerated the time for junior engineers to meaningfully contribute.

A small minority, about 4–5% of leaders and 5% of ICs, report that AI has slowed juniors down, possibly in environments where new engineers learn to generate code without learning to understand it.



Atlassian DX research supports this broader pattern. The Time to 10th PR, an onboarding signal measuring how the median number of days, dropped to 30 days in Q2 2026—more than half since Q1 2024, before the use of AI was widespread in the industry.

The AI dividend: reinvestment, not extraction.

IF IMPLEMENTATION IS GETTING CHEAPER, THE ABILITY TO CHOOSE THE RIGHT THING TO IMPLEMENT GETS MORE EXPENSIVE.

Much of the current industry conversation frames AI's capacity gains in terms of what they might replace. What engineering leaders themselves describe is different: Given a hypothetical doubling of engineering capacity, they turn their focus to investing further in quality, trust, and what they can build next.

How teams would spend a doubled capacity

If AI doubled your engineering teams' capacity tomorrow, what would you do with it?	Leaders	ICs
Invest more in quality and testing	65%	62%
Take on more projects	50%	38%
Prioritize reducing tech debt	46%	53%
Expand into new product areas	46%	33%
Reduce dependency on contractors / backfill hiring	27%	16%

Quality and testing is clearly the top choice across both leaders and engineers doing the work: The software must get better before the organization attempts to do more.

The next priority, however, diverges between leaders and ICs. For leaders, taking on new projects and expanding into new product areas are top priorities, equal to reducing technical debt.

On the other hand, ICs clearly prioritize reinvesting AI dividends in the software itself—quality, testing, and then reducing tech debt.

Where the two groups once again agree is what is least important: reducing capacity.



Atlassian DX research reinforces this opinion. The innovation ratio—the allocation of engineering effort spent on new feature work versus maintenance, toil, and operational overhead—has remained flat in the last 12 months. This suggests that gains from AI adoption are being reinvested into software quality and engineering operations, rather than new feature work.

The time saved today: research, quality, and depth

Where engineers spend time saved by AI	Leader
Doing additional research and discovery	64%
Adding additional code quality and security checks	53%
Up-leveling engineering skills	52%
Improving work-life balance	45%
Addressing tech debt	38%
Contributing to team knowledge base	37%
Contributing to areas of the business outside software development	22%

Engineers are not using their AI-saved time to ship more code. They are using it to think more, learn more, and shore up the parts of the system that were previously under-invested.

Research and discovery, quality checks, skill development, tech debt, knowledge sharing — every top use of AI-saved time is an investment in the durability of the engineering system of work, not a raw output multiplier.



Capacity expectations are real, and rising

87% of leaders expect AI to increase their engineering organization's output capacity in the next 12 months. Only **9%** expect no meaningful change; **4%** expect capacity to decrease. The reinvestment story is not the story of leaders who don't believe AI will change output. It's the story of leaders who believe it will change output and have chosen to spend that gain on the software and the people who build it.

What this means

Three implications sit inside the AI dividend data that point to a single conclusion: The organizations that treat AI as a cost-cutting event will spend the next two years looking cheaper.

The organizations that treat it as a quality and judgment investment will spend the next two years looking better.

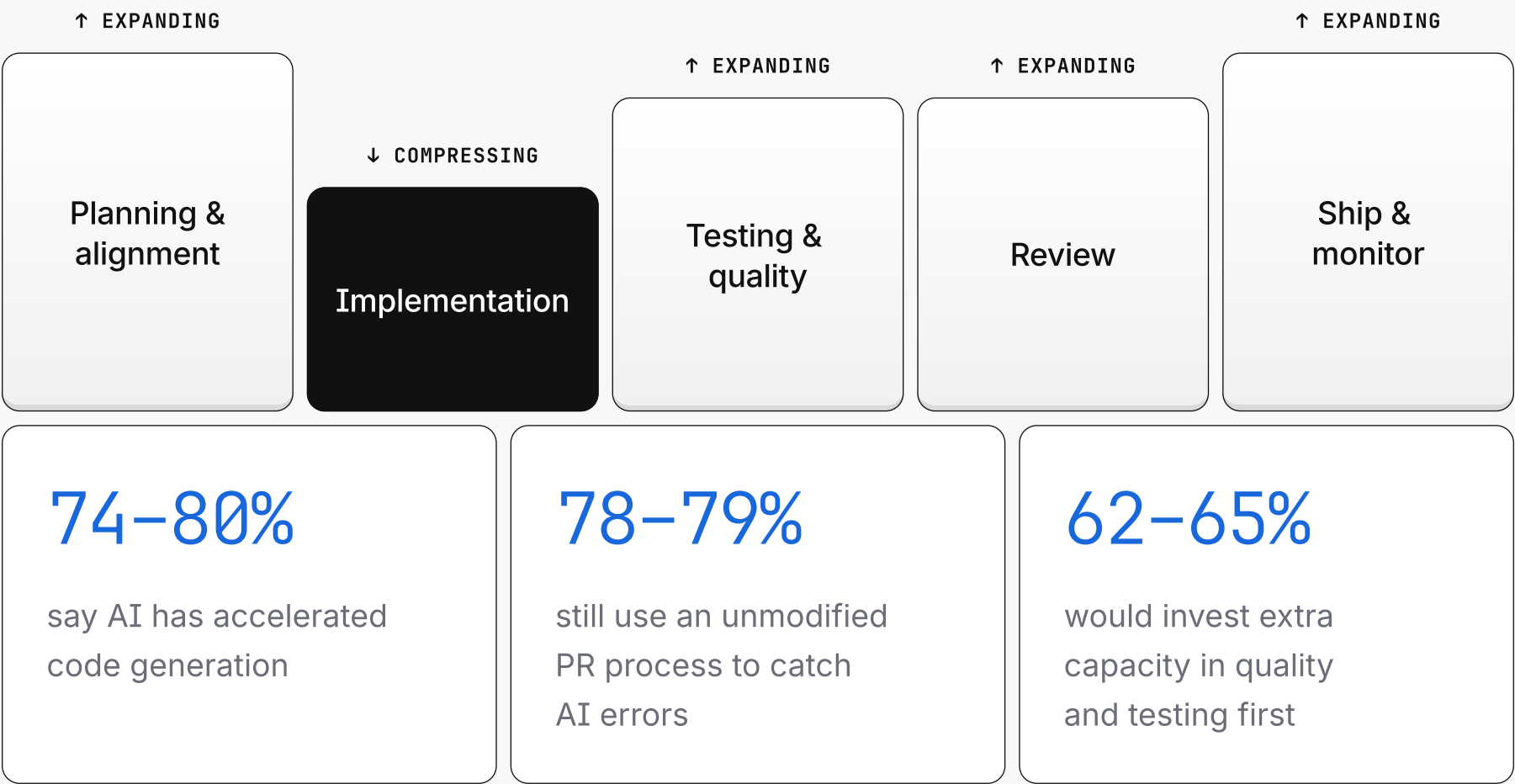
- 1 External commentary and internal leadership are describing different AI moments.** External discussion tends to describe AI as a cost lever pointed at engineering budgets. Internal leadership describes it as a quality and depth lever pointed at the parts of the software that were previously starved of investment.
- 2 Quality investment is a leading indicator of governance investment.** The 65% who would spend doubled capacity on quality and testing overlap directly with the 88% who say a governed system of record matters. The leaders reinvesting the dividend into quality are the same leaders trying to build the trust and governance infrastructure Section 02 described.
- 3 Internal value and external value may accrue on different timelines.** Increased AI utilization may generate real engineering gains, such faster cycles, fewer defects, deeper test coverage, without immediately corresponding to more features shipped or revenue recognized. Organizations measuring AI impact purely through output velocity or cost reduction may misread the signal entirely.

The remapped Software Development Lifecycle.

AI DIDN'T JUST ACCELERATE THE SOFTWARE DEVELOPMENT LIFECYCLE. IT RESHAPED IT.

The traditional SDLC was built as a sequence: plan, build, test, review, ship. Each stage had an owner, a time budget, and a handoff to the next. The underlying assumption was that implementation was the most expensive phase.

That assumption is breaking. CI/CD and cloud environments already tightened the loop between shipping and planning. AI compresses it further, but unevenly. Implementation the phase engineering has spent most of its history optimizing—is shrinking. Everything around it is expanding.



Planning & alignment	When prototyping is cheap, prototyping the right things and declaring intent become more important.
Testing & quality	More code at higher velocity means more surface area to verify—and leaders would spend their AI dividend on quality first.

Review 78% of teams are still using a review process built for human-authored code at human speed. Only 15% of ICs are very confident they can reconstruct AI-assisted reasoning six months later—meaning review isn't just gatekeeping, it's the moment where reasoning gets captured or lost.

Ship & monitor Teams are shipping more and AI-accelerated development means more code reaching production. The result is a larger production footprint to observe, against monitoring practices that are largely unchanged from the pre-AI era.

How each stage of the SDLC is shifting

Stage	Previous flow	What we see today	Where it's heading	Key data
Planning & alignment	Teams discussed requirements, wrote RFCs, and aligned before building.	Designers and Product Managers build prototypes before validating. AI accelerates research and discovery, but alignment structures haven't been redesigned to match.	Spec-driven development takes the stage, moving validation loops earlier and more frequently. Specs are no longer static plans, but the primary mechanism for declaring intent so teams and agents know what to build.	82% of leaders say AI accelerates research; 64% spend saved time on discovery.
Implementation	Traditionally seen as the most important phase, and talent was prized for the ability to implement code.	The most compressed phase. Code generation, debugging, and documentation are the top areas of AI impact.	Single-agent workflows evolve toward multi-agent orchestration across longer time horizons.	74% report accelerated code generation; 69% accelerated debugging; 56% already run agentic workflows.
Testing & quality	A distinct phase, often squeezed by deadlines. Manual test writing, periodic security reviews.	Teams say quality is their top capacity priority, but automated guardrails remain a minority practice.	Testing shifts focus towards designing evaluation frameworks, validation loops, and LLM-as-judge patterns.	65% would invest extra capacity in quality first; only 35% use automated AI-specific guardrails today.
Review	Line-by-line PR review. Designed for human-authored code at human speed. AI docs written after the fact.	Nearly 6 in 10 teams catch AI errors through an unmodified PR process. Reasoning behind decisions is being lost as fast as code is being generated.	Review evolves from diff inspection to intent and risk assessment, and captures reasoning as a byproduct of doing the work. Teams will use AI code review agents even more.	76% use unmodified peer review; only 15% of ICs (25% of leaders) very confident they can reconstruct AI reasoning after 6 months. 84% say a system of record matters; 19% have one.
Ship & monitor	Periodic or continuously deployed releases. Monitoring is considered a separate concern.	Teams deploy more but observe with the same tools they had before AI.	Monitoring feeds back into rapid iteration. Longer-running SRE agents scan production systems, gather context and data, and report incidents instantly through SRE communication channels.	87% of leaders expect capacity gains in the next 12 months. Agent visibility remains a minority practice.

AI usage is not AI readiness: Three moves to navigate the AI SDLC pivot

In the next 12 months, the engineering organizations that pull ahead won't be the ones with the most AI adoption, since experimental adoption is nearly universal now. They'll be the ones who close the widest gap in this report: the **88%** of leaders who believe they need a governed engineering system of work, and the **19%** who have actually built one. Every finding in our research is a version of that same gap. Implementation shrunk. Planning, testing, review, and ship-and-monitor all expanded around it.

Here are three moves to close that gap:

- 1 Make intent and context explicit before code is generated.**
As implementation costs shift, ambiguous intent and missing context become the cost. Teams need to specs, architectural guardrails, and test expectations as first-class inputs to development. This is the shift toward spec-driven development: making intent explicit enough that humans and agents can work from the same source of truth.
- 2 Govern and orchestrate planned and unplanned engineering loops.** The SDLC was never a straight line. Planned work moves from roadmap to spec, implementation, review, release, and learning. Unplanned work moves through incidents, test failures, security findings, customer escalations, flaky builds, performance regressions, and production alerts. As agents take on more work inside these loops, teams need clear ways to route tasks and decisions, carry context across handoffs, define autonomy boundaries, and escalate when human judgment is needed.

Here are three moves
to close that gap:

3 Make agent work traceable and observable.

With 36% of engineering organizations already run agentic workflows, teams need to understand what agents are doing in real time and why they made previous decisions. That means seeing which agents are active, what context they used, where they are blocked, and where humans need to step in; capturing a continuous trace from the initial prompt through code review, deployment, and production signals.

In the next 12 months:

Experimental AI adoption in software development was the simpler part.

The next, long-term competitive advantage won't come from even faster adoption speed, but from the system of work built around AI: connected context, embedded agent workflows, and full visibility that let humans and agents build together without losing trust in what hits production.

Methodology

This report is based on original research conducted by Prolific on behalf of Atlassian in Q2 2026. It draws on responses from more than 1,100 software professionals across two independent surveys:

Engineering leaders (n=364): Engineering Managers (71%), Heads of Engineering (9%), Directors of Engineering (7%), VPs of Engineering (1%), CTOs (7%), and CEOs/Founders (5%). Organizations range from under ten employees (12%) to over five thousand (21%). 55% Europe, 41% North America, remainder across Asia, Latin America, and Australia.	Individual contributors (n=750): Working software engineers across a similar distribution of organization sizes and regions.
---	---

All data is self-reported and reflects perception rather than measured behavior. Some questions allowed multiple selections, so percentages for those items exceed 100%.