



+

+

+

AI Code Review for Modern Engineering Teams

+

+

01	Introduction: Faster velocity, diminishing trust	03-06
02	Measuring broken trust beneath the headlines	07-08
03	Insight 1: Code is no longer the bottleneck	09-12
04	Insight 2: AI-introduced issues accumulate if they're left unchecked	13-15
05	Insight 3: As software gets more complex, it's more expensive to change	16-18
06	Conclusion: Validation restores the trust that velocity costs	19
07	Footnotes/sources:	20-22
08	Appendix	23-25

Introduction: Faster velocity, diminishing trust

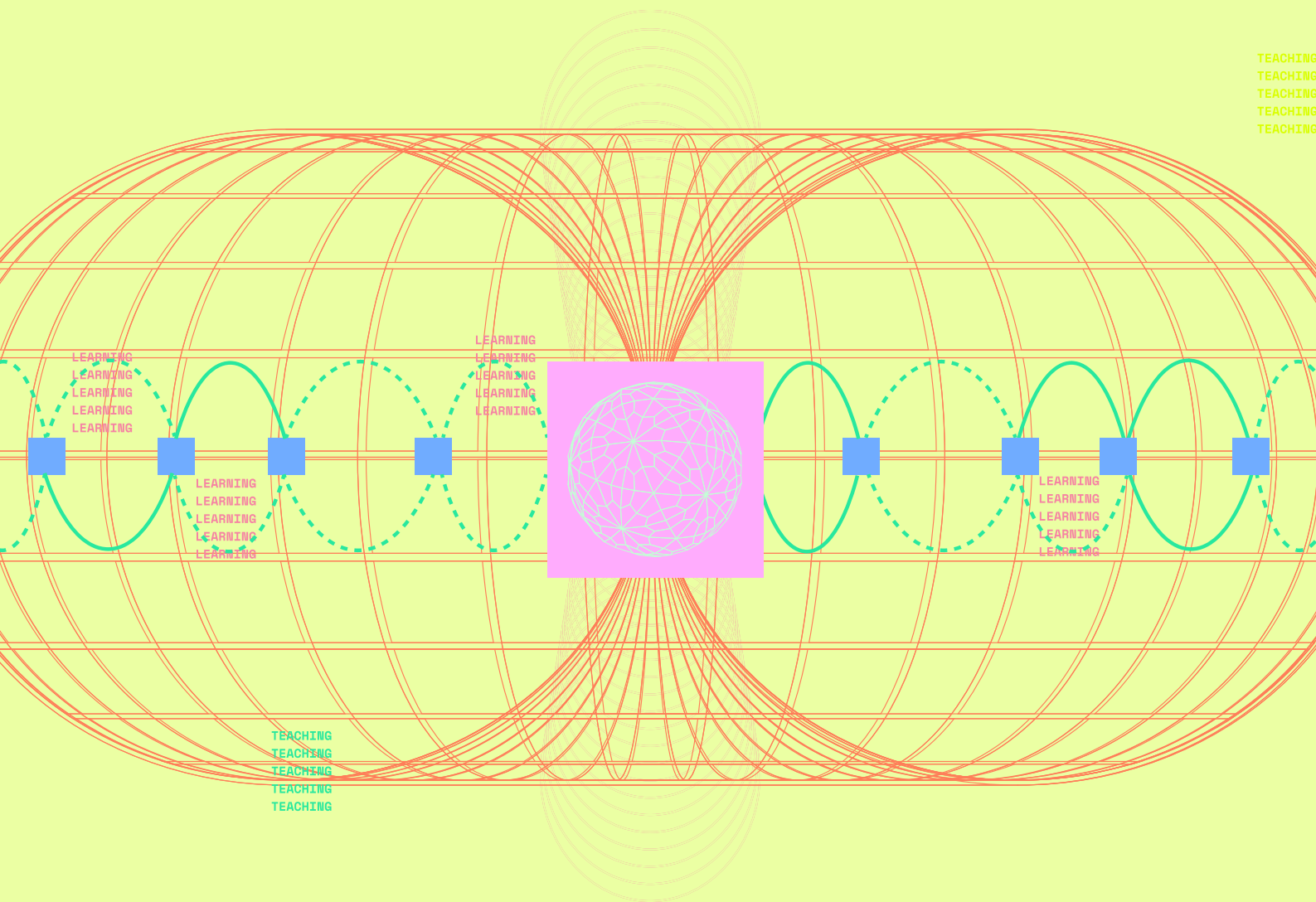
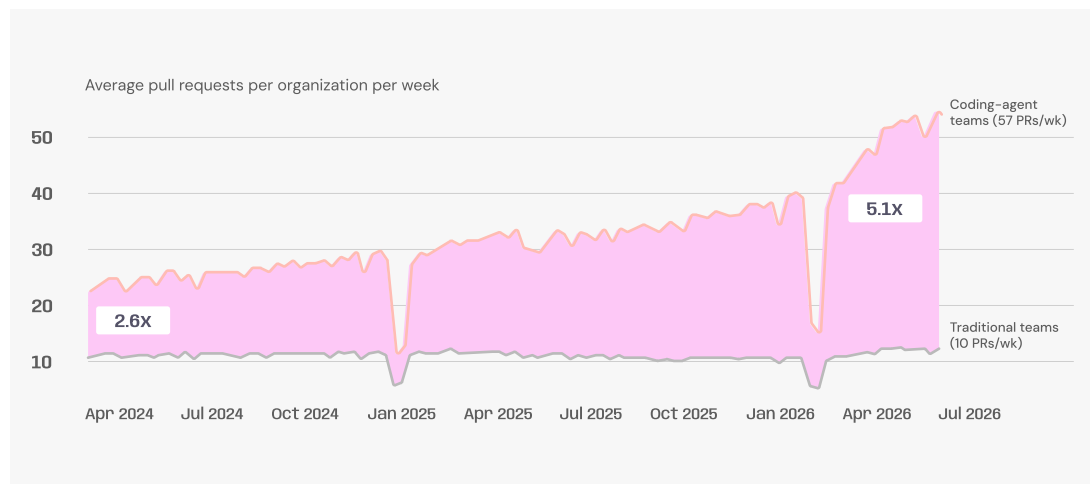
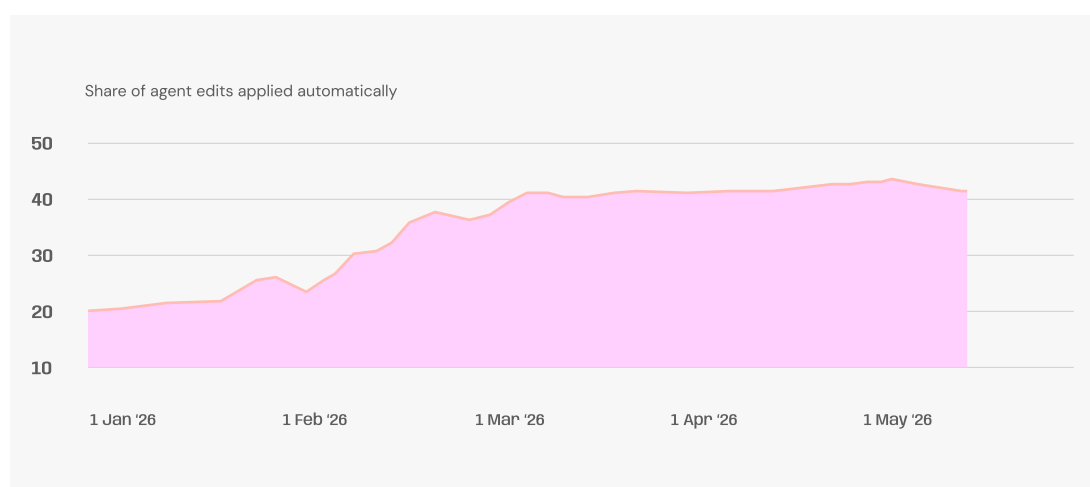


Fig.1
Linear data



Source: Linear, via The Pragmatic Engineer [1]

Fig.2
Cursor data



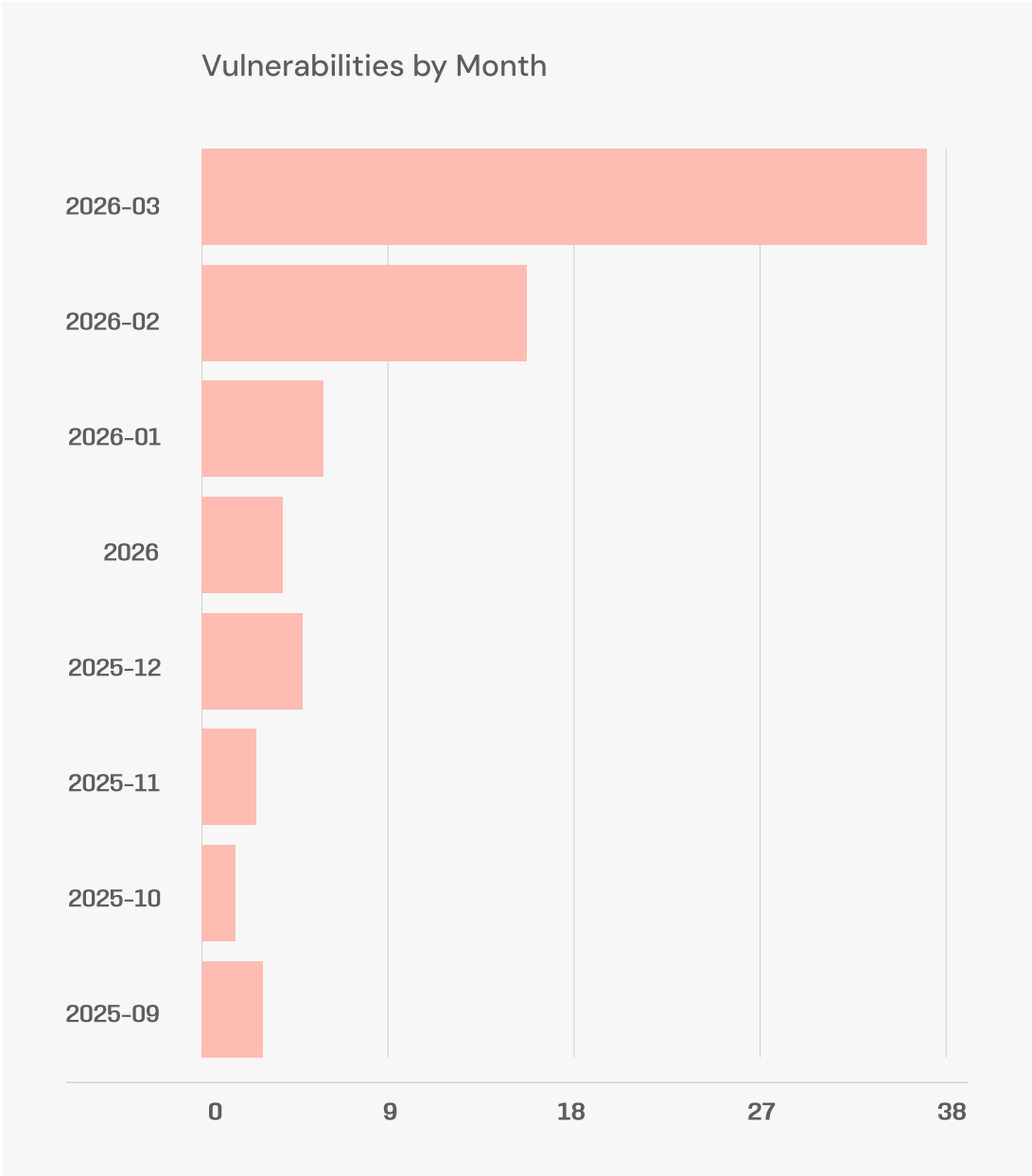
Source: Cursor Insights [2]

Engineering teams are shipping faster than ever with the assistance of AI. According to data from Linear, teams using coding agents merge five times more pull requests per week than teams that don't (Figure 1) [1].

At the same time, according to data from Cursor, the share of AI agent-generated changes reaching commits without a separate manual accept step has increased more than 5x since the start of the year, suggesting developers are trusting agents to carry more of the commit flow (Figure 2) [2].

The code is moving fast, and it's breaking trust. Software is becoming more complex and brittle.

Figure 3:
Georgia Tech
Vibe Security
Radar study



Source: Georgia Tech Vibe Security Radar [3]

78	8	43	46,831
AI-linked CVEs	AI tools	Critical/High	Advisories scanned (24% with fix)

A Georgia Tech study tracked AI-introduced security vulnerabilities across tens of thousands of security advisories. In the second half of 2025, they found 18 cases; on average, roughly, three per month. By March 2026, they found 35 in a single month (Figure 3) [3].

Brittle software comes with real business costs. At Meta, engineers confirmed to Gergely Orosz that AI-generated, AI-reviewed code (combined with security teams depleted by layoffs and reassignments) contributed to a security failure that exposed over 20,000 Instagram accounts [4], including those belonging to the Obama White House, a US Space Force official, and Sephora. Meta's CISO resigned the following day [5].

The engineering organization was never just producing code; every time the team deployed code safely, it also produced trust. When these deployments break, so does trust. "We're accumulating code faster than we're accumulating trust," Kent Beck wrote [6]. He elaborated [7], "With code, you need to trust it. You need to understand it. We don't have time to do that right now." Beck outlined three possible responses. Teams need to:

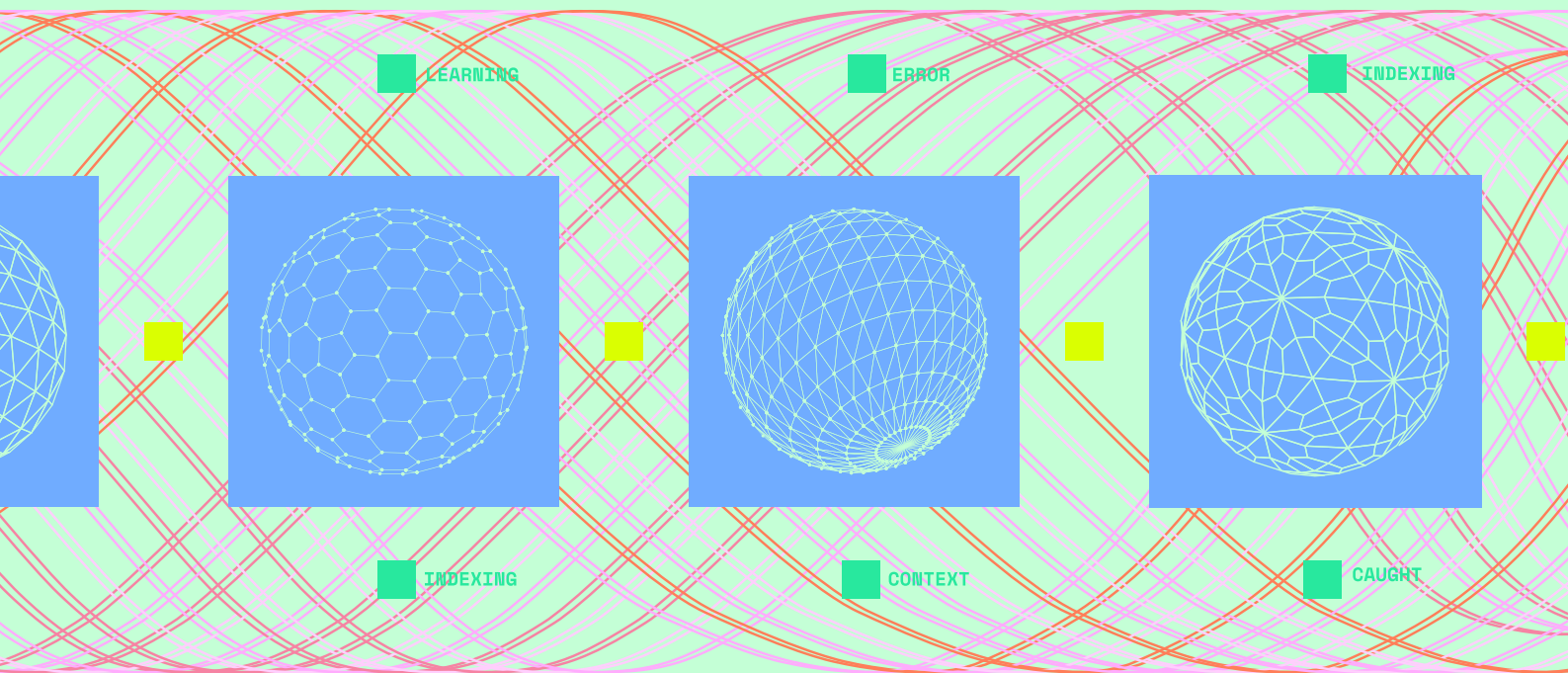
- **Accept operating without trust** in programs and the engineers who ship them.

- **Slow down code production deliberately** to match the pace of trust.

- **Learn to accumulate trust faster.**

This report dives deep into the third path, supported by academic research and interviews with engineering leaders who are accelerating trust accumulation along with their software velocity.

Measuring broken trust beneath the headlines



AI-related coding failures rarely make headlines, as most companies have no reason to publicize them.

To understand how widespread this problem really is, we turned to four independent academic research papers that conducted studies based on thousands of repositories, not just the failures companies couldn't hide:

105,364

AI-introduced issues still unresolved in the studied repositories
302,000+ AI-authored commits tracked across 6,000+ repos built with five popular coding assistants (Copilot, Claude, Cursor, Gemini, Devin), identified via Git commit metadata Source: Liu et al. [9]

+41%

more complexity

Code complexity increases after Cursor adoption across 806 GitHub repos studied. Speed gains are short-lived; quality costs continue to rise
Source: He et al. [8]

37.3%

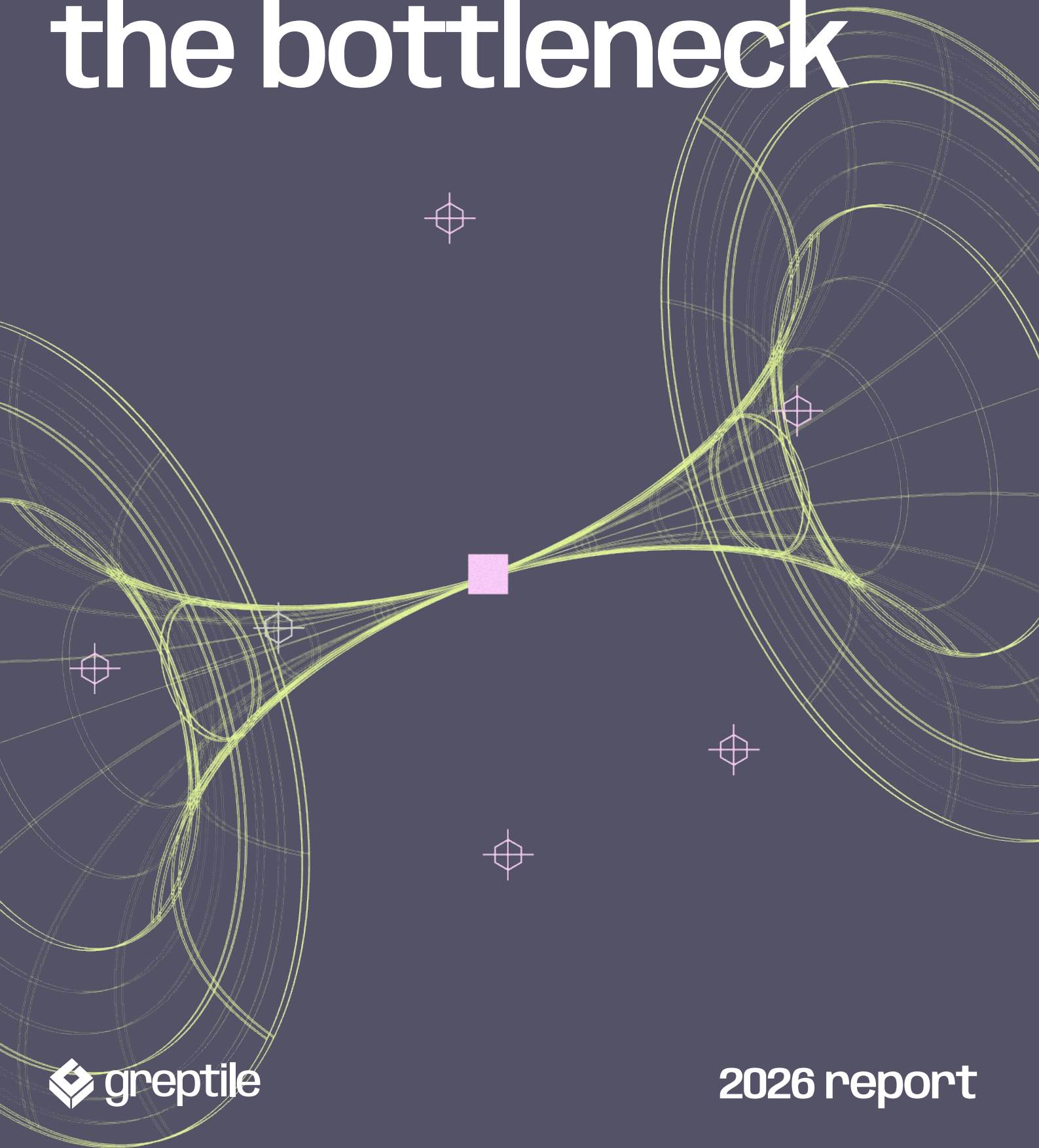
Of bugs in Claude Code, Codex, and Gemini CLI stem from API, integration, or configuration errors
3,864 confirmed bugs analyzed
Source: Zhang et al. [10]

77%

Of agent trajectories show code quality eroding the longer agents work on it. Prompting harder doesn't fix it
Measured in controlled benchmark setting, against 473 human-written repos as baseline
Source: Orlanski et al. [11]

These academic studies help establish the scale of AI-introduced risk. (Please see the Appendix for the full summary.) The following findings highlight key insights, and what engineering leaders can do about them.

Insight 1: Code is no longer the bottleneck



He et al.'s study of Cursor adoption in 806 GitHub repos found that velocity gains fade within two months of adoption. Meanwhile, code complexity (+41%) and static analysis warnings (+30%) rise at adoption and remain elevated for the full observation period [8].

The initial velocity gain overshadows the complexity that accumulates over time (Figure 4). This complexity is the new constraint, which is a matter of software validation.

As engineering constraints shift, so do the responsibilities of engineering leadership. The challenge is no longer speeding up code output, but designing engineering systems that allow velocity and trust to scale together.

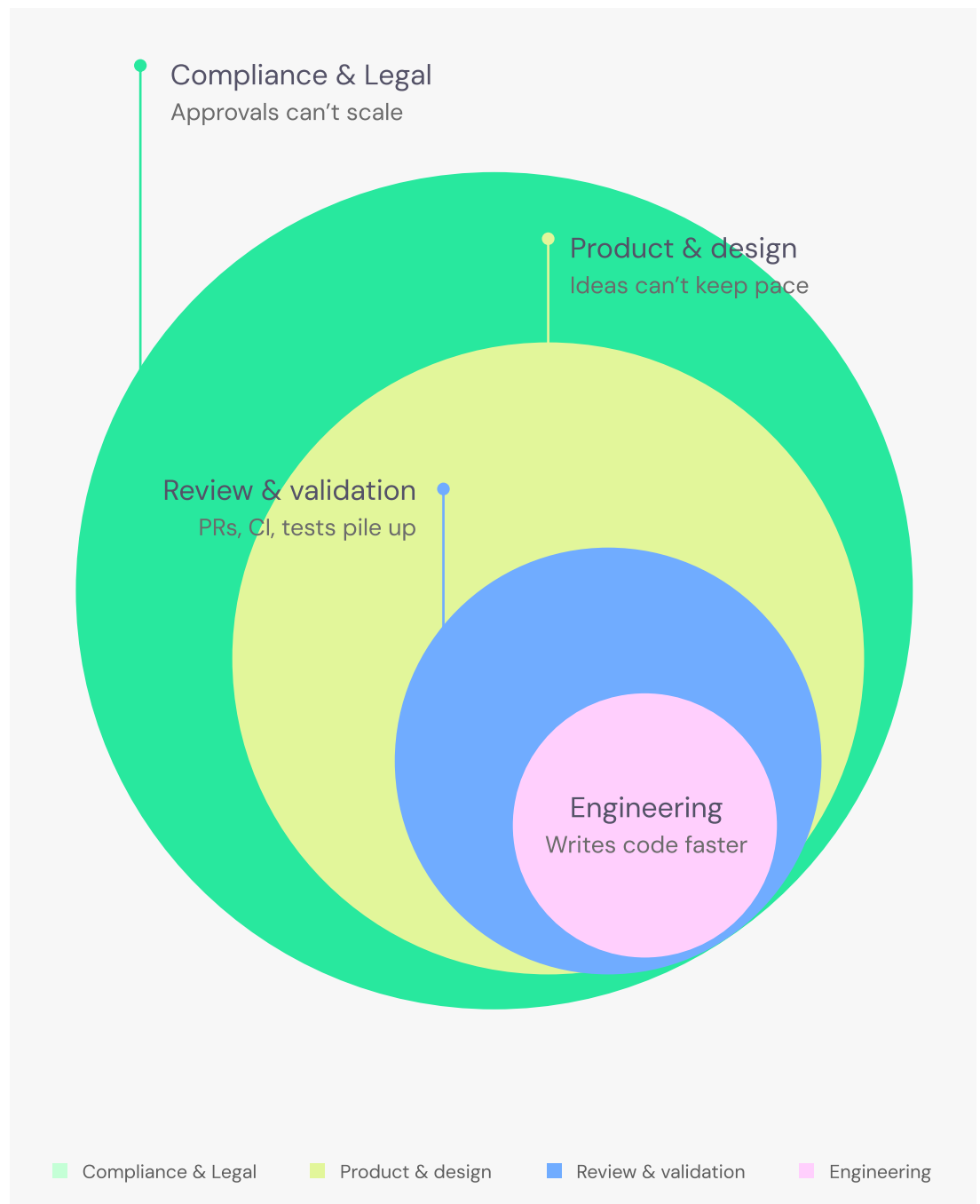
Fig.4
Speed at the Cost
of Quality diagram



Source: He et al. [8]

Coinbase's head of Wallet Chintan Turakhia thinks of this shift in constraints in concentric circles. Each inner circle used to be a bottleneck; as it accelerates, the constraint gets pushed to an outer circle. Here's an illustration of his mental model (Figure 5) [12]:

Figure 5:
Concentric circles
of constraints

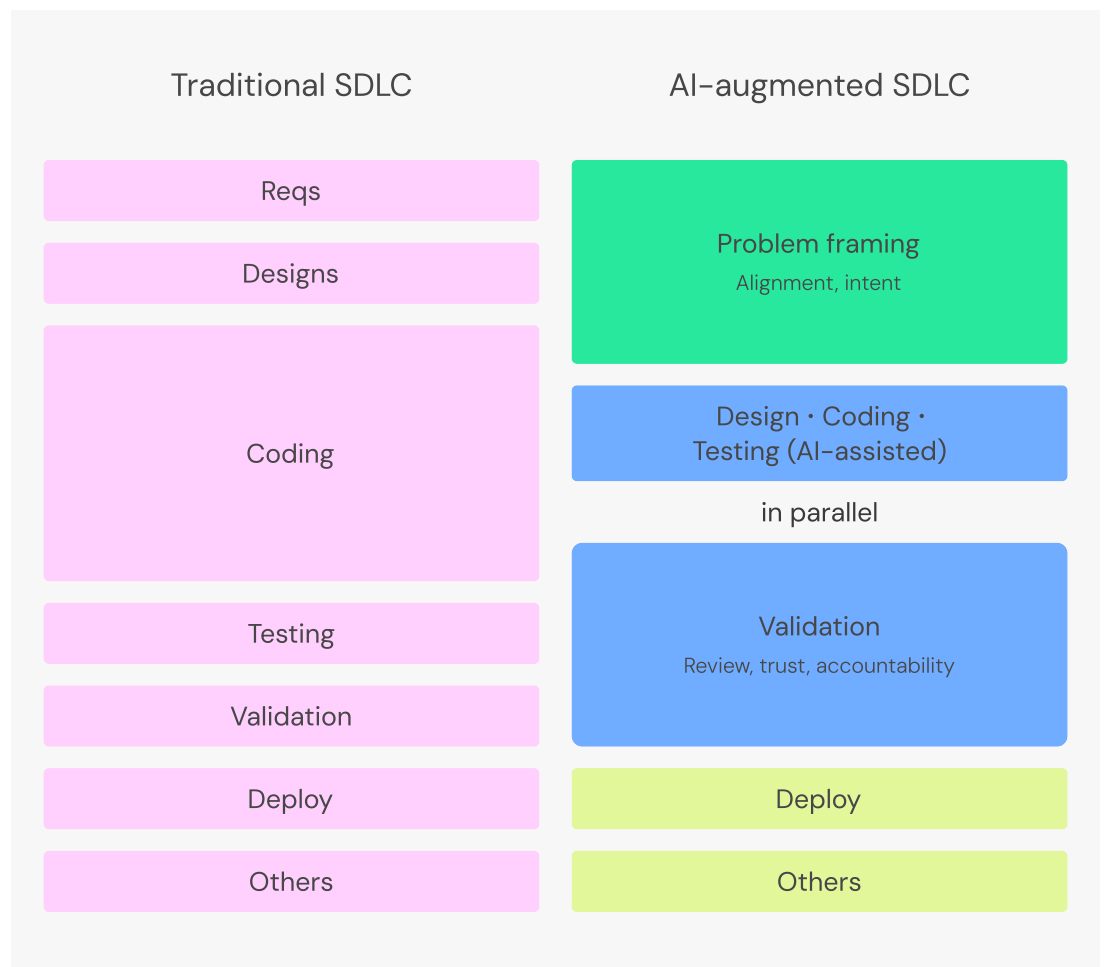


Once engineering teams can move faster, review and validation constrains the process. Then once validation can keep up with software velocity, there won't be enough product ideas and design resources. Eventually, once that keeps up, compliance and legal remains the final constraint.

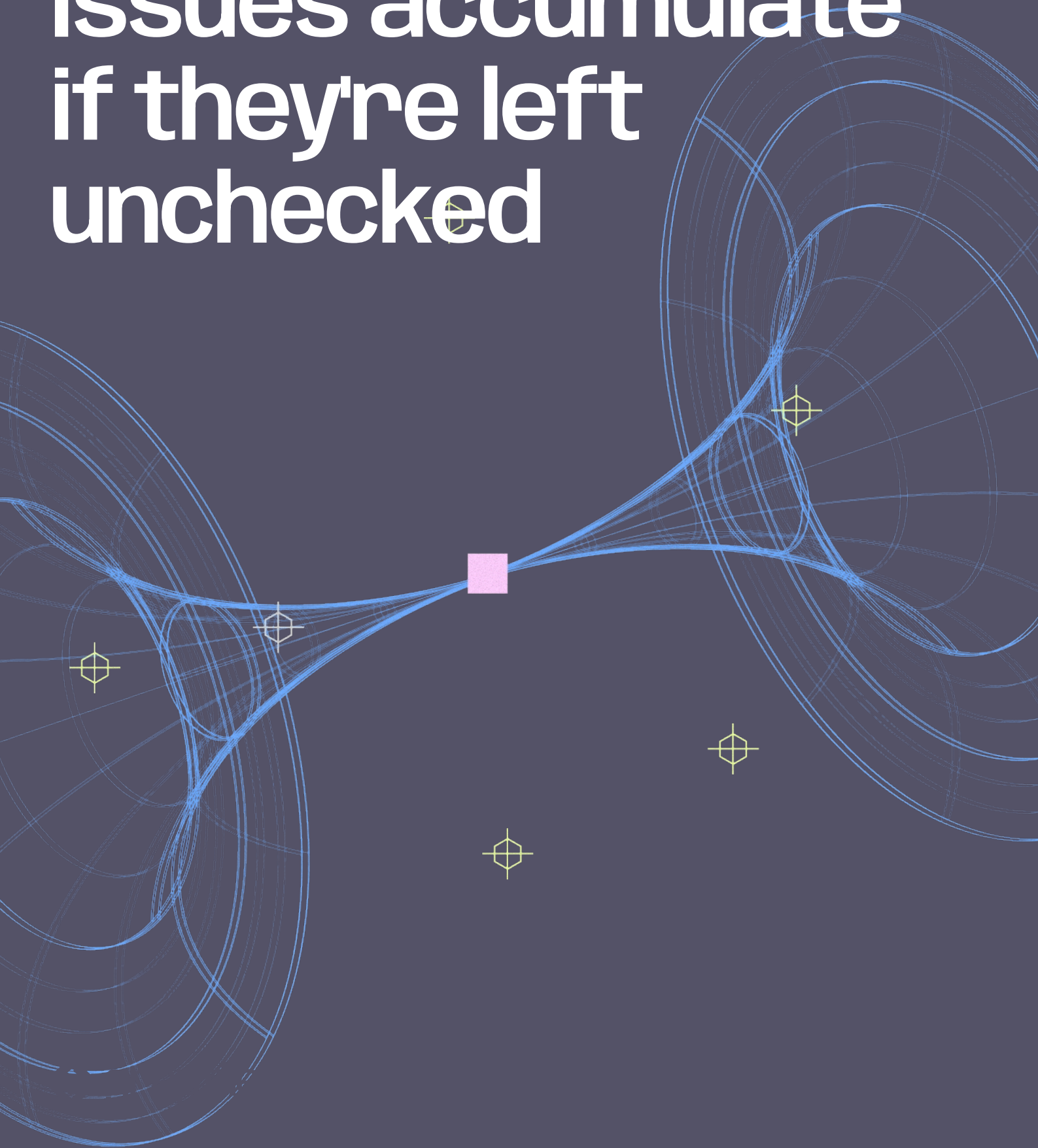
Until recently, engineering teams spent most of their time writing code. AI has sped that process up. At Atlassian, an engineering organization of about 6,000 people, coding is now roughly 15% of how they spend their time; the rest of their time is spent on new bottlenecks, including problem framing, validation, and alignment [13].

Engineering teams that build trust will spend less of their time writing code, and more time in problem framing and validation. (Figure 6)

Figure 6:
Reallocation
of time



Insight 2: AI-introduced issues accumulate if they're left unchecked

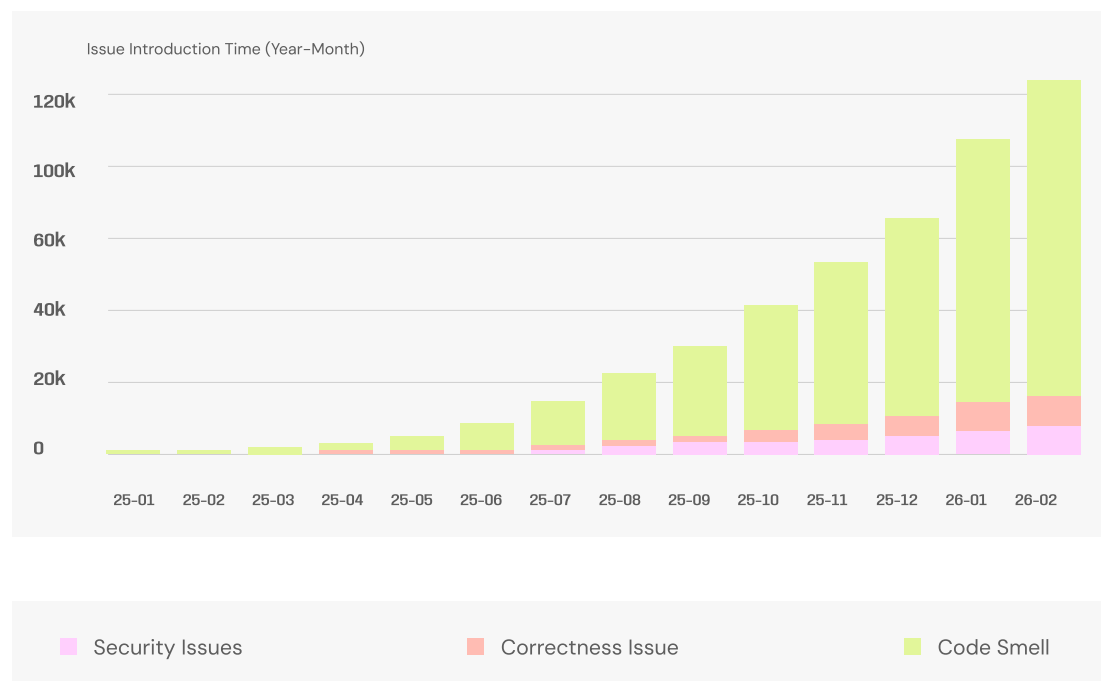


Liu et al. tracked AI-introduced issues, including code smells, correctness, and security issues, across over 6,000 GitHub repositories from early 2025 through February 2026. Overall, out of 464,900 tracked AI-introduced issues, 105,364 still survive in the version of the codebase in February 2026, corresponding to a survival rate of 22.7% (Figure 7) [9]. Each of these issues erodes trust.

Accumulating trust more quickly means building the validation infrastructure to catch issues before they ship and getting to a cumulative decrease of AI-introduced issues, not just shipping more code faster. The closer these issues get to zero, the more trust gets restored.

While individual engineers can improve review quality, only engineering leaders can establish the processes, tooling, and incentives that make trusted velocity repeatable across an organization. As AI increases the throughput of code for software engineers, the ability to scale validation becomes more important.

Fig.7 Debt Behind the AI Boom diagram



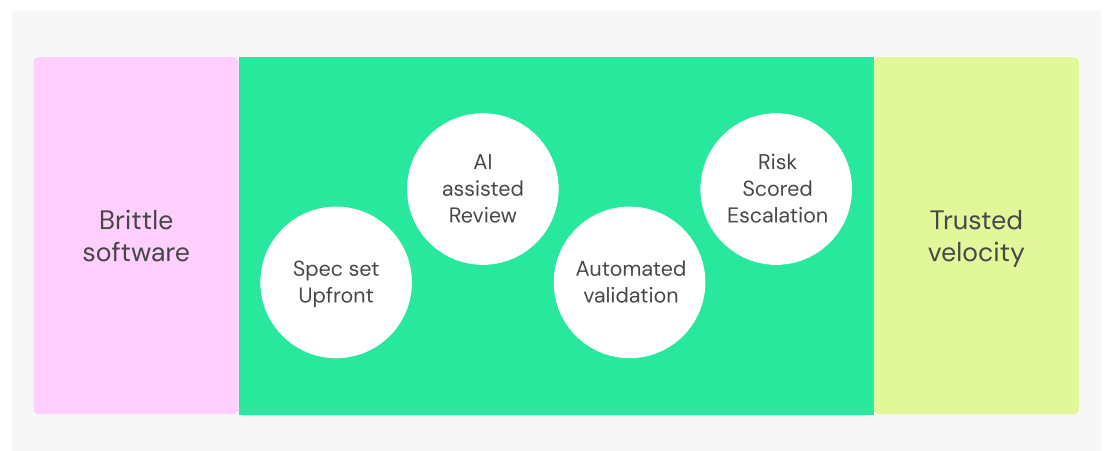
Every line of code introduces risk. As Coinbase’s Chintan Turakhia says, as the amount of code being contributed goes up, the safety guardrails need to go up with it. At Coinbase, every PR is scored against a security risk framework that governs whether an agent can approve it alone or whether it needs multiple human reviewers (Figure 8) [12]. That’s what the path from brittle software to trusted velocity, and fewer issues, looks like in practice.

Uber built similar infrastructure independently. Their internal Code Inbox tool was created specifically because, as Principal Engineer Ty Smith says [14], “Developers are spending more and more time in code review because there’s so much more code being generated.” The risk of letting bugs through increases as review fatigue sets in.

Code Inbox routes PRs intelligently. It surfaces only actionable reviews, assigning the most relevant reviewer based on ownership, availability, and timezone, and escalating automatically if SLOs are missed. Its separate Risk Profiles feature estimates the riskiness of each change and routes it accordingly.

Not all code needs the same level of scrutiny, but every line needs some.

Figure 8: Stepping stones to trusted velocity



Insight 3:

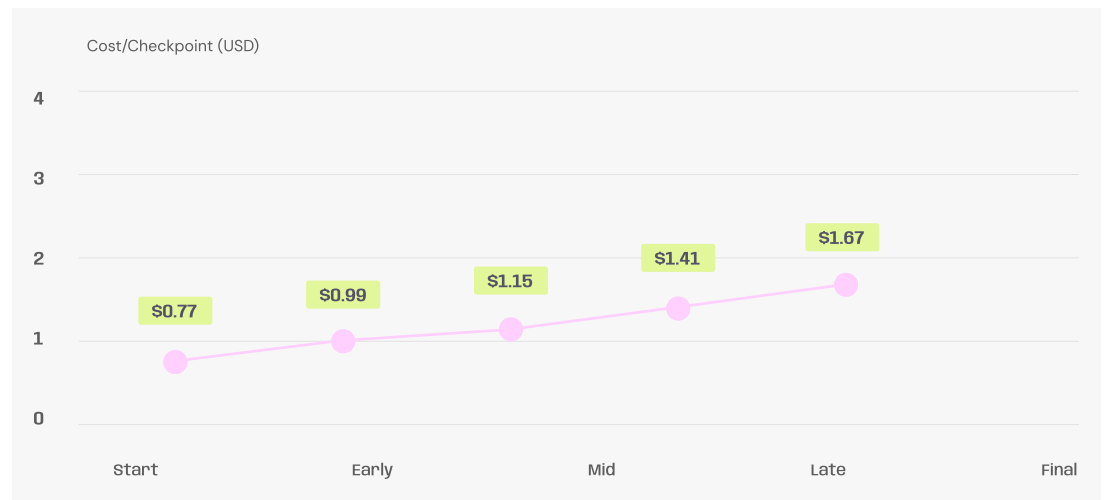
As software gets more complex, it's more expensive to change



Through their project SlopCodeBench, conducted in a controlled environment, Orlanski et al. found that the longer agents work on a codebase without validation, the more expensive every subsequent change becomes (Figure 9) [11]. These costs increase because the codebase itself becomes progressively harder to work with.

Simpler code is more trustworthy, and less expensive, code. Validation is about keeping the codebase simple enough to trust.

Fig.9 Orlanski et al. [11]

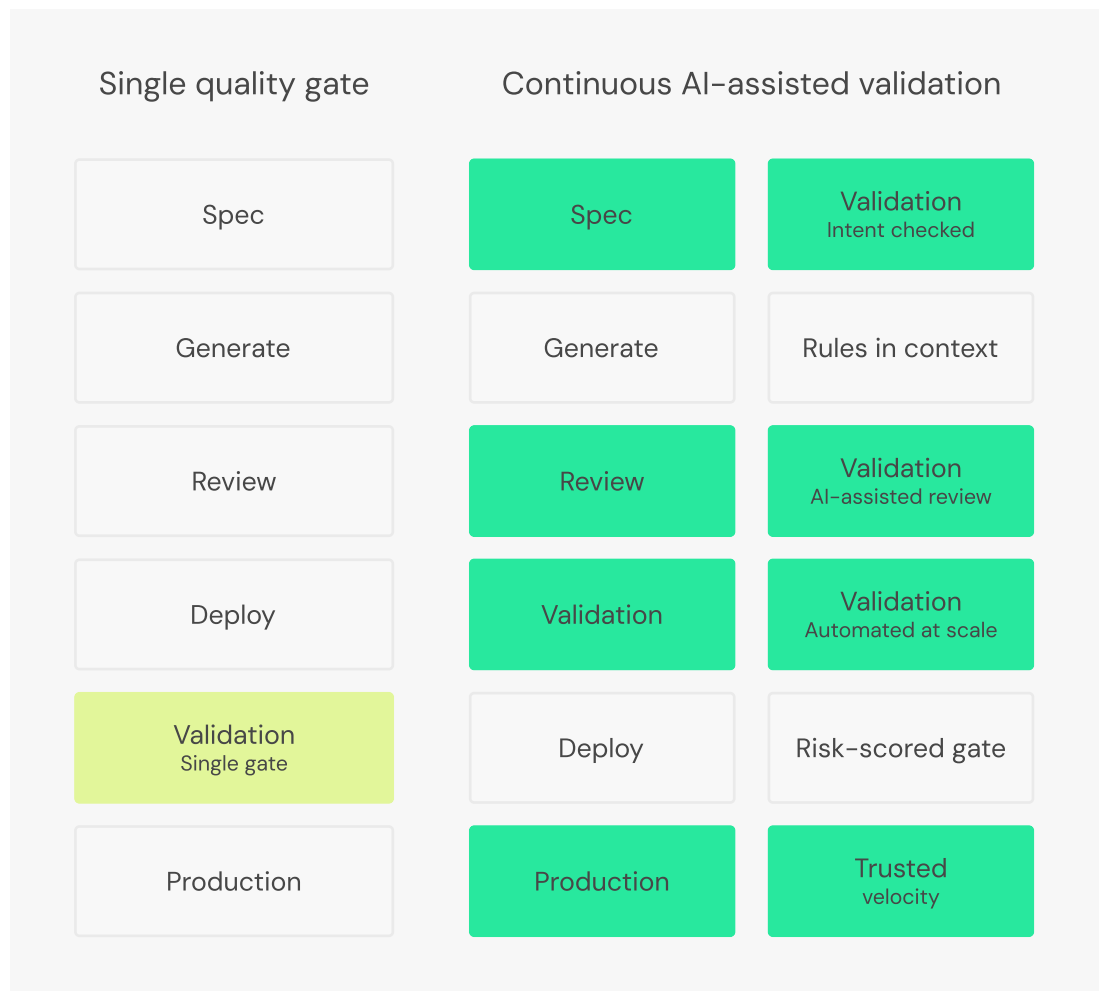


In a traditional SDLC, validation resembles a single checkpoint before production that catches what slipped through everything else. By the time a bug reaches that gate, it is expensive to fix and potentially already eroding trust. Shifting left is moving that validation effort forward earlier in the SDLC.

AI-assisted validation changes this. As Chris Kasten, Intuit’s chief architect and SVP engineering says [13], “We’re shifting left so much that the agents have all of our architecture rules, all of our security requirements fed into the context.” Review can be AI-assisted at every PR; in that sense, each PR serves as its own checkpoint. (Figure 10)

The outcome of this process is bugs are caught earlier, when they are cheapest to fix, and the codebase stays simple enough to trust. Making subsequent changes should be just as easy as early ones.

Fig.10 Shifting left diagram



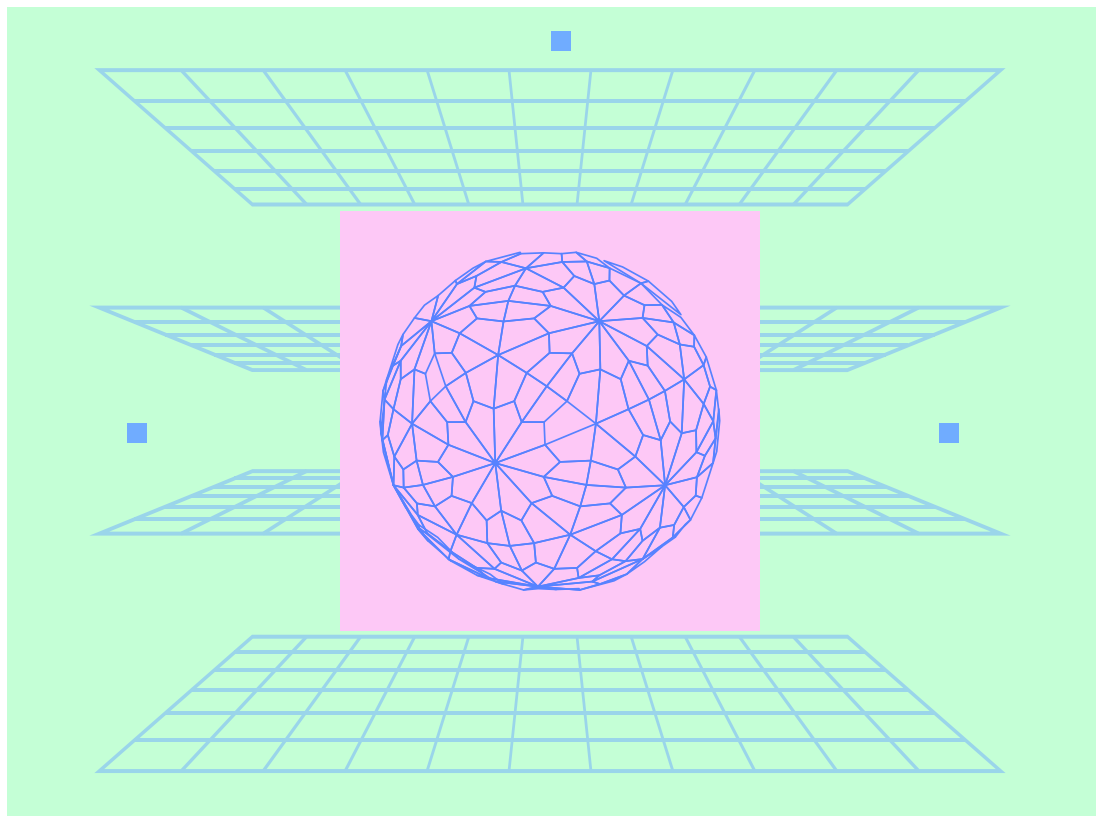
Conclusion

As Coinbase's Chintan Turakhia says [12], "AI is not an excuse to ship slop." Making sure code is reviewed thoroughly also means making the resulting software higher quality.

Velocity and validation aren't competing priorities. If velocity accelerates without validation, validation simply becomes the bottleneck. Treated as a sequence rather than a tradeoff, they help engineering teams accelerate the production of trust.

"Trustworthy behavior resulting in trustworthy software resulting in trust is the path to effective augmented development," Kent Beck writes [6]. The path to trusted velocity runs through four practices: spec upfront, AI-augmented review, automated validation at scale, and risk-scored escalation. Each one is a step toward accumulating trust more quickly.

In a world where anyone can ship code quickly, the leaders who can also ship trust will ship better products, faster. The engineering leaders in this report demonstrate that trusted velocity is achievable not by slowing down, but by building the validation infrastructure that makes speed sustainable.

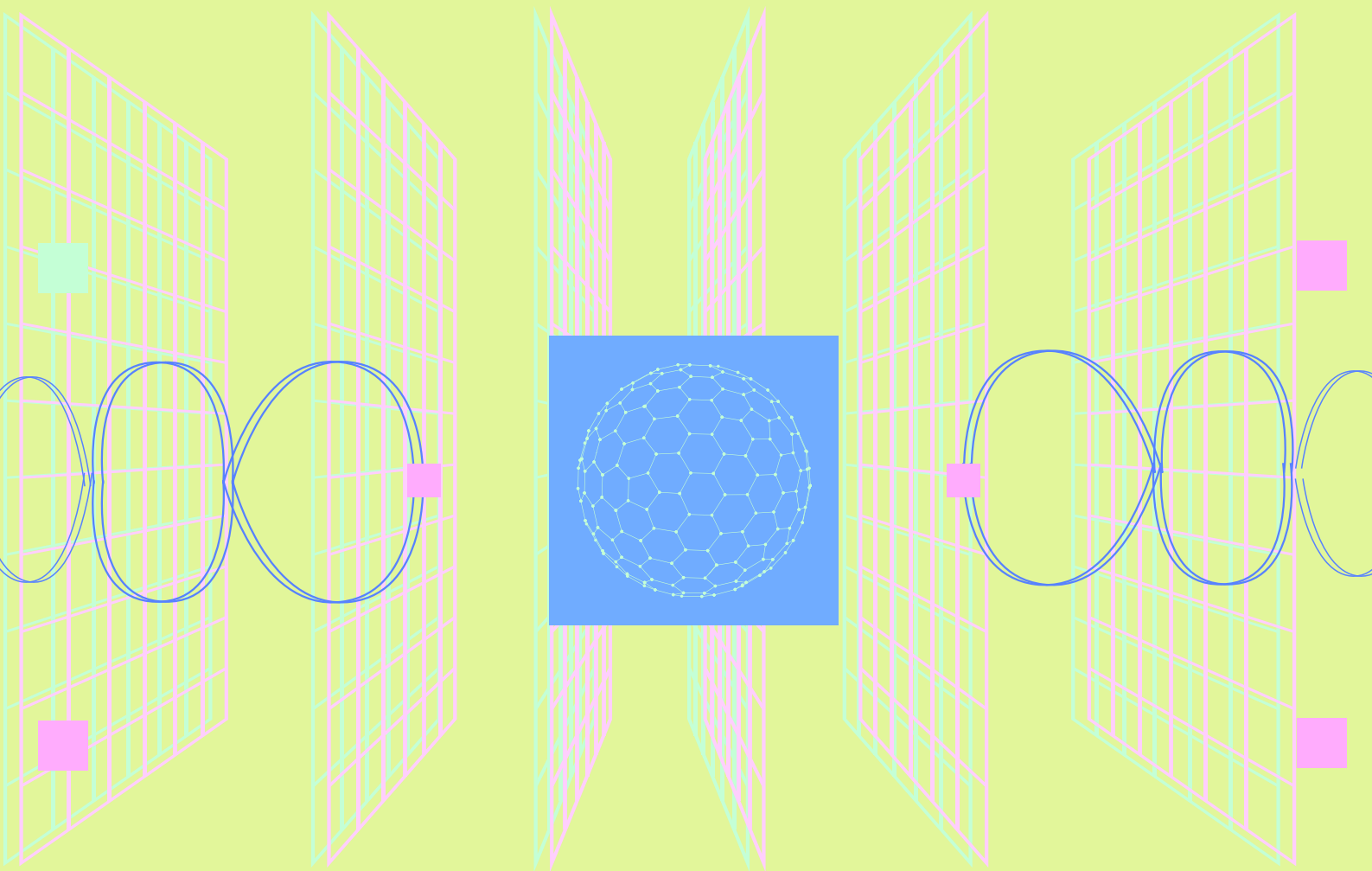


Footnotes/ Sources

- [1] Linear pull request data via Gergely Orosz, "Slow Down to Speed Up," The Pragmatic Engineer, June 2026. newsletter.pragmaticengineer.com/p/slow-down-to-speed-up
- [2] Cursor commit data. cursor.com/insights
- [3] Georgia Tech Vibe Security Radar. research.gatech.edu/bad-vibes-ai-generated-code-vulnerable-researchers-warn.
- [4] "Hackers Tricked Meta AI Into Handing Out Access to Major Instagram Accounts," Gizmodo, 2026. gizmodo.com/hackers-tricked-meta-ai-into-handing-out-access-to-major-instagram-accounts-2000766087
- [5] Meta incident sourcing: Gergely Orosz, "Slow Down to Speed Up," The Pragmatic Engineer, June 2026. newsletter.pragmaticengineer.com/p/slow-down-to-speed-up
- [6] Kent Beck, "Trust Factory," Tidy First newsletter, 2026. <https://newsletter.kentbeck.com/p/trust-factory>
- [7] Kent Beck, as reported by Gergely Orosz at Craft Conference, Budapest, June 2026. youtu.be/5wks1W-auKY?t=2665
- [8] He, H., Miller, C., Agarwal, S., Kästner, C., and Vasilescu, B. "Speed at the Cost of Quality: How Cursor AI Increases Short-Term Velocity and Long-Term Complexity in Open-Source Projects." Mining Software Repositories '26, April 2026. cmustrudel.github.io/papers/msr2026he.pdf
- [9] Liu, Y., Widiasari, R., Zhao, Y., Irsan, I.C., Chen, J., and Lo, D. "Debt Behind the AI Boom: A Large-Scale Empirical Study of AI-Generated Code in the Wild." arXiv:2603.28592, April 2026. <https://arxiv.org/abs/2603.28592>

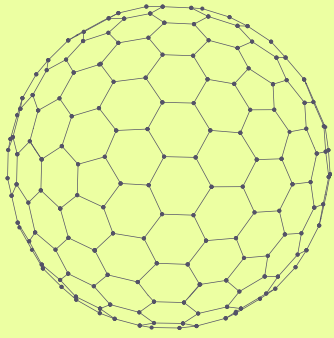
- [10] Zhang, R., Dai, W., Pham, H.V., Uddin, G., Yang, J., and Wang, S. "Engineering Pitfalls in AI Coding Tools: An Empirical Study of Bugs in Claude Code, Codex, and Gemini CLI." Foundations of Software Engineering '26, June 2026. <https://arxiv.org/abs/2603.20847>
- [11] Orlanski, G., Roy, D., Yun, A., Shin, C., Gu, A., Ge, A., Adila, D., Roberts, N., Sala, F., and Albarghouthi, A. "SlopCodeBench: Benchmarking How Coding Agents Degrade Over Long-Horizon Iterative Tasks." arXiv:2603.24755, May 2026. <https://arxiv.org/abs/2603.24755>
- [12] Chintan Turakhia. Interview conducted by Daksh Gupta, Greptile, 2026.
- [13] Taroon Mandhana, Chris Kasten, and Ryan Sokol. "How Intuit, DoorDash, and Atlassian are adopting AI coding." Cursor Compile '26, June 2026. <https://youtu.be/aF-roID9W7I?t=494>
- [14] Ty Smith, Principal Engineer, Uber. "How Uber Uses AI for Development," The Pragmatic Engineer, 2026. newsletter.pragmaticengineer.com/p/how-uber-uses-ai-for-development and <https://youtu.be/i1tZN41VKcE?t=1150>.

Appendix



Sources	What they studied	Method	Key finding	Why it matters
He et al. (CMU, 2026)	Cursor adoption in 806 GitHub repos	Difference-in-differences, matched controls	Velocity gains fade after 2 months; complexity (+41%) and warnings (+30%) persist and compound	Speed gains are short-lived; quality costs continue to rise
Liu et al. (SMU/HUST, 2026)	302,000+ AI-authored commits across over 6,000 GitHub repos built with five popular coding assistants (Copilot, Claude, Cursor, Gemini, Devin), identified via Git commit metadata	Commit-level differential static analysis, debt lifecycle tracking to latest version of the code base (HEAD)	15%+ of commits from every tool introduce at least one issue; AI introduces 1.5x more security issues than it fixes; over 22% of tracked issues still survive at HEAD	AI fixes more code smells than it introduces, but introduces more correctness and security issues than it fixes
Zhang et al. (York/Concordia, 2026)	3,800+ closed, confirmed bugs from the public GitHub issue trackers of Claude Code, Codex, and Gemini CLI	Open-coding taxonomy across bug type, root cause, symptom, and architectural layer	67% of bugs are functional; 37.3% stem from API/integration/config errors rather than model reasoning; bugs concentrate in tool orchestration and command execution layers	Most failures happen at system boundaries (e.g., API mismatches, config errors, environment incompatibilities) not inside the model's reasoning.

Sources	What they studied	Method	Key finding	Why it matters
Orlanski et al. (Wisconsin/MIT/Washington State, 2026)	15 AI coding agents across 36 iterative software development problems spanning 196 checkpoints, in a controlled benchmark setting	Agents extended their own prior code across checkpoints; structural erosion and verbosity measured at each step against 473 human-written Python repos as a baseline	Complexity concentrates in a shrinking number of functions and redundant code accumulates with each iteration. This happened in 77% of agent trajectories.	In this controlled benchmark setting, AI-generated code starts lower quality and gets worse the longer agents work on it. Prompting harder doesn't fix it



AI Code Review for Modern Engineering Teams

