undo

RESEARCH REPORT

Overcoming the limitations of coding agents in complex software systems

Contents

INTRODUCTION

THE MODERN ERA OF SOFTWARE ENGINEERING

It doesn't matter whether engineers see AI as a threat to job security or a force multiplier that releases their full potential: coding agents are now central to the way teams build and deliver software.

Based on an independent survey of 300 engineering directors and managers leading teams that build mission-critical software on large, complex codebases, this report explores:

- How engineering time is currently split between coding, debugging, and other tasks
- Challenges and risks of using AI agents to accelerate delivery in complex codebases
- Techniques used to address the risks and build confidence in using AI coding tools
- Which AI models engineering teams favor for debugging and code comprehension
- How engineering leaders and teams weigh up cost vs. capability in model selection
- Industry-specific insights for engineering leaders

With such widespread adoption, the volume of code created is growing at an unprecedented rate. But we all know there's a catch.

The dramatic reduction of the cost of creating code has simply moved the bottlenecks downstream, where engineers struggle with code comprehension, debugging, and incident investigations. If the promised productivity gains are to be realized, AI must be applied to the software delivery cycle, not just code generation. But AI has so far proven itself unequal to effective comprehension and debugging of complex, large-scale codebases.

The difficulty of understanding why an application behaves the way it does, coupled with AI's tendency to give confident answers from incomplete evidence, makes it prone to hallucinate the cause of bugs and instabilities. That means engineers still need to step in to solve these downstream problems and it is now where they are spending most of their time.

Nowhere is the pain more acutely felt than in the organizations delivering the services and infrastructure the economy runs on, such as finance, semiconductors, and networking technology, where escaped defects or unstable codebases can have catastrophic consequences.

PART 1

WHERE TIME IS SPENT NOW

Writing code has suddenly become far cheaper, but in mission-critical codebases, that code must still be understood – engineers must still be accountable for it. The more demanding task is understanding what that code does, how it affects existing codebases, and debugging it when an application doesn't behave the way it's expected to.

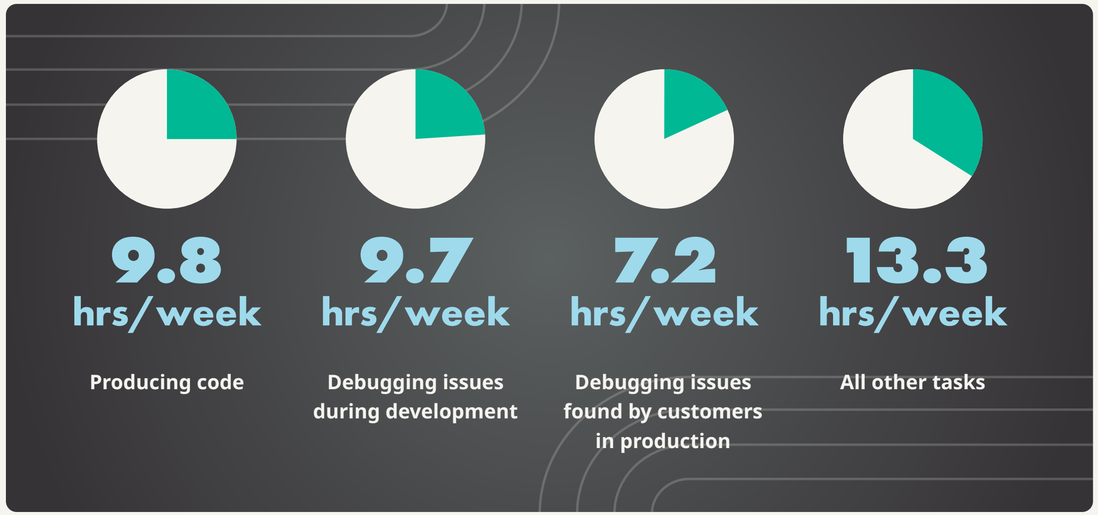
Engineers now spend nearly twice as much time debugging issues in their code as they do producing code in the first place.

42%

of engineering teams' time is spent on debugging.

Based on the assumption of an average 40-hour working week.

How engineering teams' time is divided between tasks



Nine in ten (91%) engineering leaders say code comprehension and debugging take even longer for more complex software such as C/C++ codebases. That challenge will only become more pronounced as the volume of AI generated code increases.

Code that's completely wrong isn't so much of an issue, as it's usually fairly obvious where the cause of the failure lies, and modern AIs are usually able to root-cause and fix such issues. Where engineers struggle most is with code that's almost, but not quite right. Those are the times when manual intervention is required, and days are spent tracing across a multitude of systems to unravel what went wrong and why.

PART 2

THE IMPACT OF CODING AGENTS ON RELEASE CYCLES

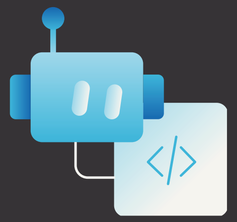
Coding agents promise faster software delivery, but unless AI is applied across the entire software development lifecycle, then the bottleneck simply shifts downstream.

While agents can generate swathes of code, engineers of mission-critical systems still need time to comprehend it and assume accountability for it.

Engineering leaders are in an impossible bind. Huge productivity gains are expected in return for the significant sums being spent on AI; and yet quality and accountability must be maintained.

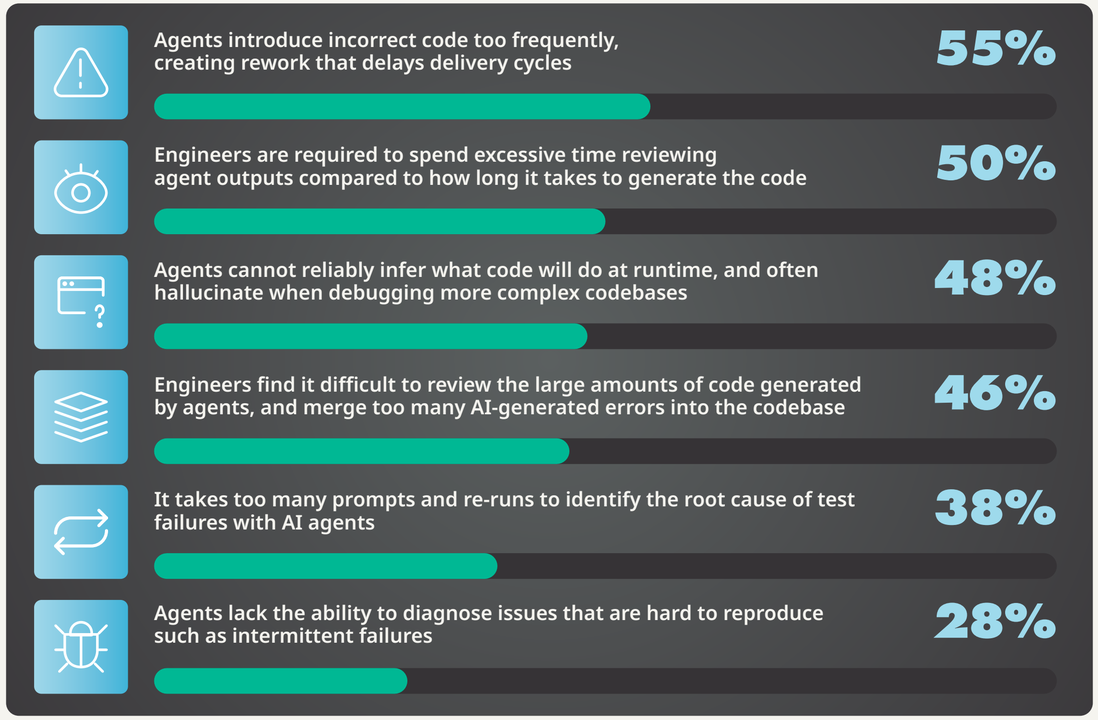
35%

of AI generated code has not been fully comprehended before engineering teams push it to production.



Now that AI is generating most of the code being produced, engineers no longer have the inherent understanding they used to. That makes it easier for defects to escape, and when something inevitably goes wrong, nobody has the knowledge to trace the failure back to its root cause.

Challenges engineering teams face in accelerating software delivery with AI agents



Issues organizations have encountered as a result of their use of AI coding tools

Loss of productivity due to engineers needing to ‘unpick’ AI-generated code

Budget overruns due to agents needing to re-run tasks multiple times to reach the correct output

Incorrect diagnosis of the root cause of an issue due to a hallucination

Test escapes, serious defects, or poorly optimized code entering production

Production incident / service outage affecting internal users or customers

- **Multiple times per month**
- **At least once in the past six months**

79% **of engineering leaders say despite being able to produce code much faster, release cycles are no faster than before.**

The issues introduced by the use of AI coding agents are most acute in the more complex environments that power mission-critical industries.

Trading platforms, network operating systems, and applications of a similar nature run on legacy codebases with subtle interdependencies that are poorly documented. Detailed ‘institutional knowledge’ is undocumented, sitting only within engineers’ heads and so the AI inevitably makes subtle mistakes. But so much code is being generated that it is difficult for engineers to review and understand it properly.

It’s no wonder that so few enterprises have achieved a meaningful improvement in release cycles.

As the volume of AI-generated code increases, review fatigue will creep up for engineering teams, and the challenges they’ve experienced already will only worsen. The risk is that this will lead to more code pushed to production before engineers have comprehended it, more escaped defects, and more unresolved tickets, and further gaps in accountability.

Ultimately, the productivity gains promised by AI will remain limited unless engineering teams can also apply it effectively to comprehension and debugging. If they fail to do so, the time engineers save in code generation is simply absorbed by slower investigation and debugging, or lost later through production incidents and security failures.

PART 3

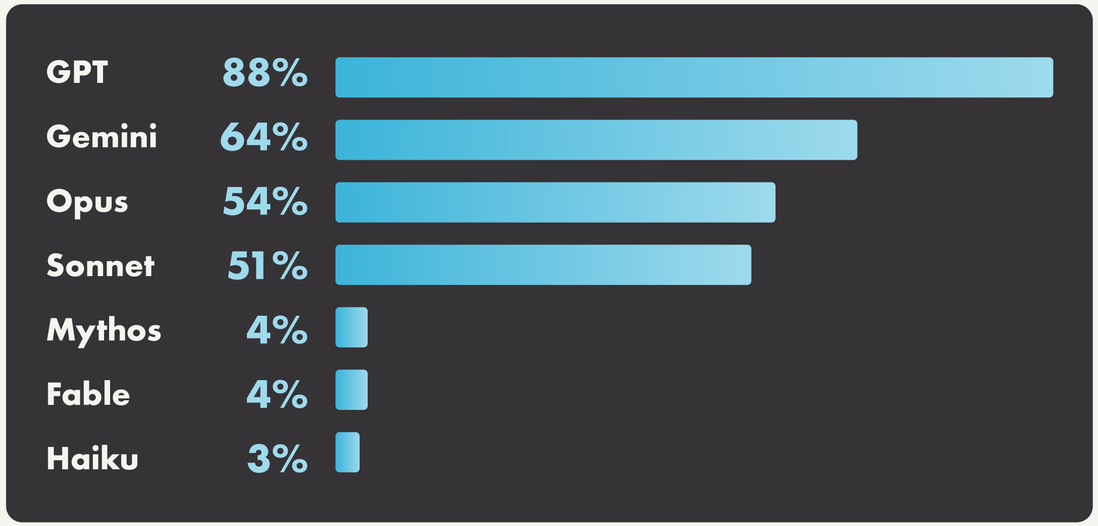
CALCULATING THE TRUE VALUE OF AI

Measuring lines of code generated only tells the beginning of the story about the impact of AI on engineering productivity.

To get the real story of productivity gains from coding agents, it's important to look at metrics like **defect rate** and **mean time to resolution**. Engineers should apply AI in downstream workflows to bring those numbers down, as well as optimizing their coding practices.

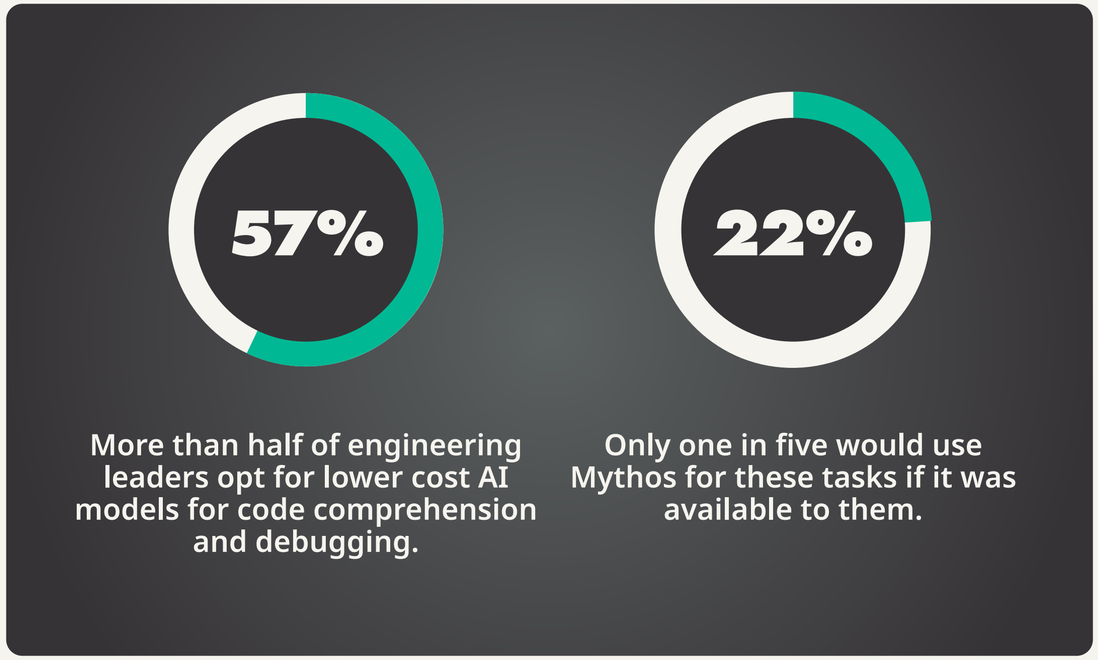
Different models have their individual strengths and weaknesses, and there are trade-offs between capability and economy for any given task. Engineering teams use a whole host of models for debugging and comprehension, but GPT and Gemini ride high on their list.

AI model families engineering teams use to understand and debug complex codebases



Few engineering teams are using the most capable models for code comprehension and debugging, even in more complex environments. **Nearly two-thirds (63%)** of leaders say the cost is prohibitive to authorize the use of higher tier AI models for this work. This is perhaps surprising given that this task now represents the main bottleneck in software delivery. It may be that a lack of confidence in the returns from AI investment steers leaders towards controlling spending more carefully than they might have otherwise.

Rising costs are a sizable barrier to increasing the use of more powerful AI in downstream delivery.



82%

of respondents think most enterprises will be priced out of Mythos or Fable-level AI models.

There's also a degree of cynicism about the impact the planned IPOs of Anthropic and OpenAI will have on the trajectory of token costs in future. **Four in five (82%)** engineering leaders are concerned that the costs of coding agents will go through the roof as the labs prioritize making Wall Street happy.

To scale AI without breaking the budget, engineering leaders need to find a way to close the gap between lower cost models and those at the frontier. That only becomes possible by acknowledging that the power of an AI model is only as good as the **context** it can read – especially when it comes to work like code comprehension and debugging.

PART 4

PROBLEM-SOLVING IN COMPLEX CODEBASES

More than a third (37%) of teams use AI agents for comprehension and debugging only for straightforward codebases.

When they do use AI for code comprehension and debugging in more complex codebases, engineering teams rely on a range of workarounds to give a greater degree of confidence.

Techniques organizations use to mitigate the challenges and risks of using AI agents for code comprehension and debugging

80%

of engineering leaders say coding agents struggle to solve difficult problems in large-scale, complex codebases.

66%

Using a human in the loop to approve any AI agent action

55%

Enriching source code analysis and logs with specific context about what happened at runtime

48%

Developing hypotheses and then testing them on agents rather than asking the agent to identify the cause/solution

40%

Using agents to check the work of other agents in autonomous workflows

HOW UNDO CAN HELP

82%

say AI agents would be far more useful for code comprehension and debugging if they were grounded in the context of what happened at runtime.

Relying on human effort to lead AI to the solution for every bug or the cause of unexpected behavior is unsustainable given the pace at which agents are generating code. That's why it's critical to give agents all the evidence and data they need to reach **accurate conclusions** independently.

Engineering leaders almost unanimously agree with that principle. **More than four in five (82%)** say AI agents would be far more useful for code comprehension and debugging if they were grounded in the context of what happened at runtime.

The most effective way to provide that context is to give the agent a recording of program runtime behavior, so it can see everything code did on execution and identify the precise moment something went wrong.

Not only does that solve AI's accuracy problem, but it also addresses its cost issues. Given a complete recording of runtime execution, more affordable AI models can understand codebases and solve the issues they contain with far greater accuracy than a frontier model that only has source code to go on – at a fraction of the cost.[1]

Taking this approach, engineering leaders can build agentic workflows that are far more useful for solving the hardest issues that delay releases and slow progress – especially in the most complex codebases.

1 CppCon 2026 conference poster: comparison data on how runtime context improves the economics of AI

About UNDO

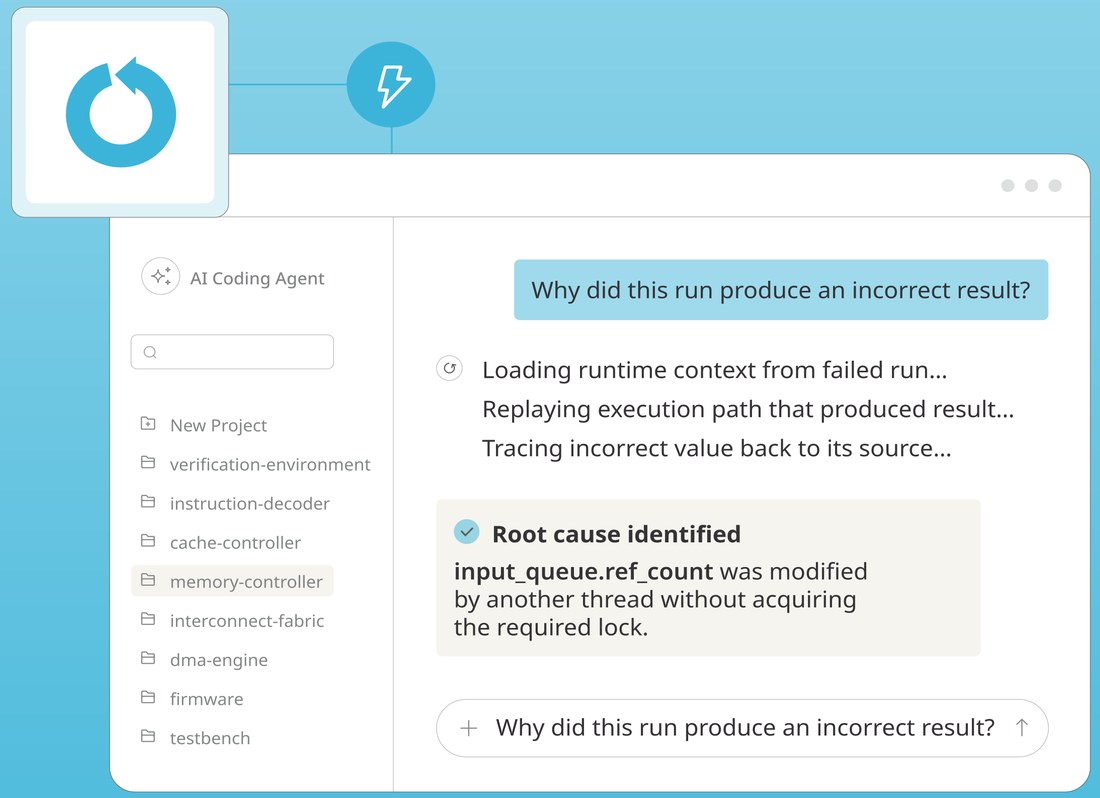
Undo enables coding agents to solve the most complex problems on the most complex codebases.

By providing a deterministic, self-contained, portable recording of complete program execution, Undo enables engineering teams to give their agents the runtime context they need to reason about dynamic application behavior in the same way they already reason about static code.

The result: fully automated root-cause analysis.

With Undo, developers have been able to solve problems up to 100x faster in some of the world's most demanding software environments including at AMD, AWS, Cisco, and Palo Alto Networks.

To find out how Undo could help your engineering team unlock the potential of AI agents, visit undo.io



METHODOLOGY

This report is based on a survey of 300 senior engineering leaders in large organizations ($250m+ revenue) that deliver mission-critical software built on large, complex codebases, including 93% working with C/C++. The fieldwork and analysis were conducted on behalf of Undo by independent research firm Coleman Parkes during July and August 2026.

Regional samples

United States (200), UK (100).

Industry samples

Financial services (30%), networking (30%), semiconductor design (11%), computational software (15%), data management (14%).

Respondent roles

Senior manager/director of software engineering/development (24%), software engineering/development manager (19%), director of engineering (17%), director of software architecture (15%), director/head of quantitative development or quant trading (10%), director/head of low latency engineering (8%), director of silicon design engineering (8%).

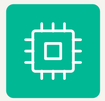
KEY INDUSTRY FINDINGS

Financial services

- **Software engineers** in financial services firms spend an average of 15.7 hours per week debugging code – accounting for 39% of their working week; the second lowest of any industry.
- **Nearly a third (31%) of AI generated code** has not been fully comprehended before engineering teams in financial services firms push it to production – although the proportion is higher in all other industries.
- **Nearly two-thirds (62%) of respondents** in financial services firms say engineers are required to spend excessive time reviewing AI agent outputs compared to how long it takes to generate the code – far more than any other sector.
- **GPT (89%) and Gemini (63%)** are the most popular AI models used for understanding and debugging more complex codebases in financial services.

Networking

- **Engineering teams** in networking companies spend an average of 19.5 hours per week debugging code – accounting for 49% of their working week; the highest in any industry.
- **More than a third (37%) of AI generated code** has not been fully comprehended before engineering teams in networking firms push it to production – the joint highest of any industry alongside semiconductor design.
- **Networking firms (68%)** are amongst the most likely to find AI coding assistants useless for debugging anything other than the most simple of codebases, behind only semiconductors (69%) – compared to the cross-industry average of 55%.
- **GPT (89%) and Gemini (68%)** are the most popular AI models for understanding and debugging more complex codebases in networking companies.



Semiconductor design

- **Software engineers** in semiconductor design firms spend an average of 16 hours per week debugging code, accounting for 40% of their time.
- **More than a third (37%) of AI generated code** has not been fully comprehended before engineering teams in semiconductor design firms push it to production – the joint highest of any industry alongside networking.
- **Semiconductor designers** face the greatest struggle with agents being unable to reliably infer what code will do at runtime and hallucinating when debugging complex codebases – with 63% of respondents identifying this as a challenge, compared to the cross-industry average of 48%.
- **Two in five (41%) engineering leaders** in the semiconductor design sector say their agents lack the ability to diagnose issues that are hard to reproduce such as intermittent failures – compared to the average across industries of 28%.
- **GPT (88%) and Gemini (69%)** are the most popular AI models for understanding and debugging more complex codebases in semiconductor design companies.

Data management

- **Software engineers** in data management dedicate 16.6 hours (41.5% of their time) to debugging tasks each week.
- **More than a third (36%) of AI generated code** has not been fully comprehended before engineering teams building data management software push it to production.
- **Nearly two-thirds (60%) of respondents** in data management firms say engineers find it difficult to review the large amounts of code generated by agents, and merge too many AI-generated errors into their codebase – compared to the cross-industry average of 46%.
- **GPT (88%) and Sonnet (62%)** are the most popular models for understanding and debugging more complex codebases in data management software companies.

Computational software

- **Software engineers** working on computational software spend 15.4 hours (38.5% of their time) on debugging tasks each week, the lowest of any industry.
- **More than a third (35%) of AI generated code** has not been fully comprehended before engineering teams building computational software push it to production.
- **Exactly half of respondents** working on computational software say it takes too many prompts and re-runs to identify the cause of test failures with AI agents – compared to the overall average of 38%.
- **GPT (85%) and Gemini (63%)** are the most popular models for understanding and debugging more complex codebases in computational software companies.

undo

undo.io