5 WAYS AI-GENERATED CODE CHURN IS KILLING YOUR ROI

//SquidSpark

The “False Velocity” Trap

Occurs when AI tools allow developers to commit large volumes of code in seconds, creating the illusion of productive output rather than quality work. Writing code becomes faster, but the real cost of software development lies in understanding, validating, and trusting that code. When teams must invest additional time to review and verify AI generated output, the perceived productivity gains no longer hold.

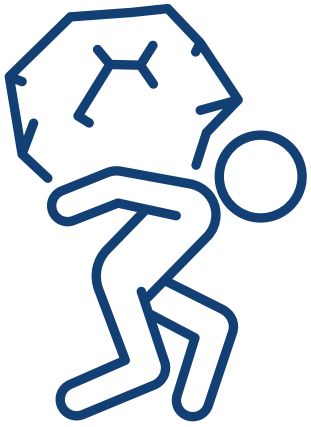
The AI Productivity Paradox (2025) shows that time saved writing code does not translate into faster delivery of the final product. Although high AI usage teams closed 21 percent more tasks on average, ***code review time also increased by 91 percent.*** This highlights that software development costs are driven primarily by the human effort required to understand and validate code.[1]

The Hidden Cost of AI-Generated Code

AI-generated code leads to a **154%** increase in pull request size and a **70%** rise in issues found in AI-generated PRs.[1] Reflecting this added burden, **45%** of developers in the 2025 Stack Overflow survey identify correcting AI suggestions as their top frustration.[2]

Escalating Technical Debt

AI-generated code often prioritizes short-term functionality over long-term maintainability and architectural consistency. It satisfies initial requirements but adds unnecessary complexity and ignores project-specific standards as systems scale. Over time, this produces a fragmented codebase with inconsistent patterns, making future changes slower, riskier, and more expensive.



The Army of Juniors: The AI Code Security Crisis report finds that architectural issues appear at significantly higher rates in AI-generated code. Two primary patterns; **by the book fixation** and **lack of refactoring**, are observed in **80% to 90% of analyzed projects.** By the book fixation produces code that follows textbook practices but fails to align with system specific architecture, while lack of refactoring leaves code rigid and difficult to maintain. These issues accelerate technical debt and increase long term maintenance costs.[3]

What a Study of 211 Million Lines of Code Reveals

GitClear's latest study shows that as AI coding tools become more widely used, code quality is declining, and code duplication is increasing. In its analysis of 211 million lines of code, GitClear finds that *technical debt is growing up to 8 times faster* than before. A key metric underlying this finding is the substantial rise in code duplication, with an eightfold increase in adjacent duplicate code blocks of five or more lines. This form of *duplication now occurs at more than ten times its 2022 rate.* API evangelist Kin Lane warns "I don't think I have ever seen so much technical debt being created in such a short period of time during my 35-year career in technology." [4]

The “Silent” Security Risk

AI models frequently suggest outdated libraries or patterns with known vulnerabilities due to training on older datasets. AI-generated code often omits standard security controls, input validation, and secure configuration, increasing the risk of data exposure and compliance violations. When these issues are identified during security audits, teams must refactor core components late in the development cycle, resulting in delays and higher engineering costs.

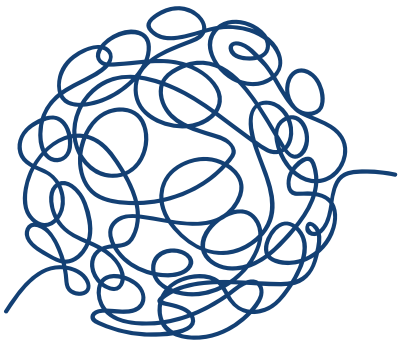
According to the Veracode 2025 GenAI Code Security Report, approximately **45% of AI-generated code contains security vulnerabilities.** Java shows the highest risk, with Python, C#, and JavaScript also affected. The report finds that LLMs choose the insecure method nearly **50%** of the time when presented with both secure and insecure options.[5] Many of these vulnerabilities stem from logic and input validation. **AI-generated code fails to prevent cross-site scripting (XSS) 86% of the time and fails to properly sanitize log inputs 88% of cases.**[6]

AI Assistance Increases Confidence, Not Security

A joint Stanford and NYU study finds that developers using AI coding assistants produce code with more security vulnerabilities than those who do not. Despite this, they report higher confidence in the correctness and security of their code. The study concludes that developers frequently overestimate the security of AI-assisted code.[5]

Integration Complexity and Bloat

AI-generated snippets often include unnecessary dependencies or overly heavy logic for tasks that could be handled more simply. The result is growing integration complexity and “dependency hell.” Applications become bloated, harder to test, and more likely to experience build failures and deployment disruptions.



Human developers identify patterns and shared components, while AI often generates code for each request independently, leading to unnecessary dependencies and duplicate logic. For example, a 50-line user authentication function can grow to 200 lines, filled with redundant error handling, unused imports, and avoidable complexity.[7] According to an October 1, 2025 analysis of GitHub data, **developers on average checked in 75% more code than they did in 2022**, indicating that teams now have substantially more code to review, test, and maintain.[8]

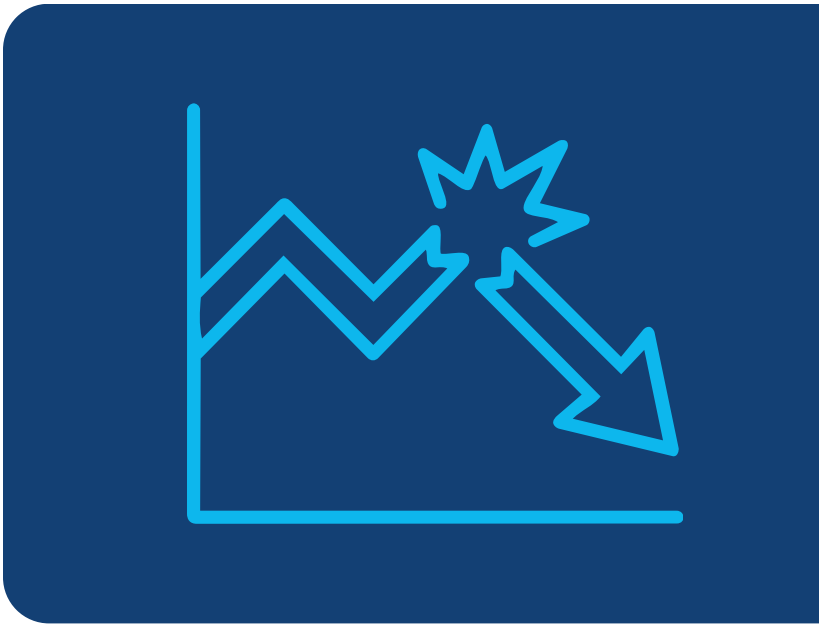
Knowledge Erosion

Relying too heavily on AI-generated code can weaken a team's understanding of its own systems. When developers deploy code without fully understanding how it works, knowledge gaps build up over time. These gaps become evident during maintenance, when the root cause is unclear and teams struggle to fix complex bugs. The result is extended downtime and stalled work as teams work through unfamiliar code.

The Ox Security report highlights a **"comments everywhere"** paradox in AI-generated code. In **90 to 100 percent of cases**, AI tools add extensive comments that make the code appear clear. However, these comments often describe what the code does rather than why it is designed that way. As a result, developers rely on surface-level explanations without fully understanding the underlying logic, which makes debugging and long-term maintenance more difficult.[3]

Rising AI Use, Falling Stability

Without a shared understanding of how systems are built, new team members take longer to become productive and teamwork becomes less effective. Google's DORA report finds that every **25% increase in AI adoption is associated with a 7.2% drop in delivery stability.** As Greg Brockman warns, "Say no to slop. Managing AI generated code at scale is an emerging problem, and will require new processes and conventions to keep code quality high." [4]



SquidSpark helps companies unlock the promise of AI

SquidSpark was founded by technology leaders who helped keep critical infrastructure running — from national cellular networks and power-generation systems to nationwide healthcare data exchange platforms that people depend on every single day. Those large-scale successes were delivered through proven frameworks, operational rigor, and hard-earned experience — principles that are embedded into the very fabric of what is SquidSpark.

At SquidSpark, AI operates inside a framework defined by the same veteran architects who have delivered these critical systems. Not only does our team know how to use AI, but we have also built AI systems.

Our team scaffolds every project upfront — defining patterns, approved dependencies, and security-by-design principles that both human and AI contributors must follow. Human-led analysis and technical design always come first, and AI is used only in controlled, reviewable blocks to improve consistency and speed — not to generate entire solutions autonomously. Each user story is reviewed and analyzed by a human developer and the approach to implement it is dictated by a human developer rather than giving AI the freedom to choose the approach.

Our development lifecycle embeds decades of best practices, including SAST, DAST, SCA, and automated testing, ensuring AI output is continuously validated within enterprise-grade guardrails.

In this way, AI becomes a disciplined accelerator within a proven delivery framework, not a replacement for engineering judgment.

With this approach, we can mitigate the 5 major risks associated with using AI in development:

False Velocity Trap
We limit AI usage to small, well-scoped implementation blocks that align with human-led architectural decisions. This ensures that acceleration occurs without introducing hidden rework that would erode delivery timelines later.

Escalating Technical Debt
Architect-defined scaffolding, strict pattern enforcement, and controlled dependency selection prevent AI from introducing unnecessary abstractions or unapproved libraries. Technical debt is addressed in real time, in manageable increments, rather than accumulating silently across large AI-generated codebases.

Silent Security Risk
Security is embedded into the framework itself, ensuring that AI-generated code inherits hardened patterns rather than inventing new ones. Continuous SAST, DAST, SCA, and automated test execution provide objective validation before code ever reaches production.

Integration Complexity & Bloat
AI operates within predefined architectural boundaries and approved tooling, eliminating uncontrolled layering and abstraction sprawl. Because generation is incremental and human-reviewed, complexity is detected and simplified immediately rather than discovered late in QA or production.

Knowledge Erosion
Human engineers lead requirements analysis, technical design, and dependency decisions, with AI assisting only in structured implementation tasks. This preserves deep system understanding, strengthens engineering judgment, and ensures expertise compounds rather than degrades over time.

Let SquidSpark help you **unlock the potential** AI can bring to your organization.

SQUIDSPARK

References

1. Mohart, N. (2025a, September 9). The AI productivity paradox: Why faster coding doesn't mean faster delivery. The AI Productivity Paradox: Why Faster Coding Doesn't Mean Faster Delivery. https://natanmohart.substack.com/p/the-ai-productivity-paradox-why-faster

2. Lessard, J. (2025, December 1). Are ai coding assistants really saving developers time?. Are AI Coding Assistants Really Saving Developers Time? https://axify.io/blog/are-ai-coding-assistants-really-saving-developers-time

3. Farry, P. (2025a, November 18). Ai-generated code creates new wave of technical debt, report finds. InfoQ. https://www.infoq.com/news/2025/11/ai-code-technical-debt/

4. Munn, J. (2025, June 16). The hidden costs of AI-assisted development, and why faster coding doesn't mean faster delivery. | by John Munn | Medium. The hidden costs of AI-assisted development, and why faster coding doesn't mean faster delivery. https://medium.com/@johnmunn/the-hidden-costs-of-ai-assisted-development-and-why-faster-coding-doesnt-mean-faster-delivery-04c22935dfd1

5. Oviedo, C. (2025, September 24). Is AI Coding really saving time & money? the numbers say "mostly yes"-with new risks to manage. Is AI Coding Really Saving Time & Money? The Numbers Say "Mostly Yes"—With New Risks to Manage. https://www.linkedin.com/pulse/ai-coding-really-saving-time-money-numbers-say-mostly-carlos-oviedo-dihvc/

6. Medium. (2026, January 12). Vibe Coding Debt: The security risks of AI-generated codebases | by InstaTunnel | Jan, 2026 | medium. Vibe Coding Debt: The Security Risks of AI-Generated Codebases. https://medium.com/@instatunnel/vibe-coding-debt-the-security-risks-of-ai-generated-codebases-7e3a038edf09

7. Why ai-generated code costs more to maintain than human-written code | by AlterSquare | Medium. Why AI-Generated Code Costs More to Maintain Than Human-Written Code. (2025, December 12). https://altersquare.medium.com/why-ai-generated-code-costs-more-to-maintain-than-human-written-code-91b57256bd6a

8. Why ai-generated code costs more to maintain than human-written code | by AlterSquare | Medium. Why AI-Generated Code Costs More to Maintain Than Human-Written Code. (2025, December 12). https://altersquare.medium.com/why-ai-generated-code-costs-more-to-maintain-than-human-written-code-91b57256bd6a