permuta
www.permuta.com

You can’t prompt your way out of Enterprise Complexity

AI Vibe coding hype is real. So are the constraints no one is discussing.

Martin Jennings, General Manager, Product Group, Permuta Technologies 17 Mar *2026*

Executive Summary

Artificial intelligence–assisted software development has emerged as one of the most widely discussed disruptive technological innovations in the modern software industry. Tools such as GitHub Copilot, Claude Code, Cursor, and similar systems are designed to generate code, tests, documentation, and system scaffolding from natural language prompts. These systems promise to significantly increase developer productivity and potentially reduce manpower requirements in software engineering organizations.

In the commercial technology sector, AI-assisted coding has quickly moved from experimental tooling to everyday development infrastructure. Major technology firms report that a meaningful percentage of their code is now produced with AI assistance. These developments have led to widespread speculation that software engineering workforces may shrink substantially as automated development tools mature.

However, the narrative surrounding AI-generated code has increasingly taken on the characteristics of a technology hype cycle. Public discussion frequently emphasizes developer productivity and manpower reduction while overlooking structural limitations, long-term maintenance costs, and operational risks. In a nutshell the hype has been focused on one aspect of the decision-making equation, manpower savings, without truly factoring all the variables that will bound and, in many cases, limit the value of a wholesale shift to AI-assisted development. Further, these nascent value calculations are based on imperfect cost data that are highly subsidized by venture capital.

These limitations are particularly important for the United States Federal Government and Department of Defense. Government systems frequently operate in secure environments, rely on large monolithic enterprise codebases, and must satisfy strict compliance, traceability, and security requirements. Many mission-critical systems operate on enterprise server stacks that must be included when making architectural design decisions. For example, Permuta leverages Microsoft Dynamics, SQL Server, and Windows Server as a platform for on-Premises solutions and the Azure Power Platform for SaaS deployments. We don’t have access to Microsoft’s underlying code base to refactor it using AI-assisted development, nor would we want to. Microsoft has spent decades developing standards-based platform solutions that not only provide reliable and secure infrastructure but also

bulk licensing discounts. The same can be said for solutions built on top of SalesForce, Palantir, and other Enterprise solution providers.

Within the Defense Industrial Base, contractors and companies supporting the Federal Government and Department of War, the promise of rapid AI-generated development faces several constraints. These include context window limitations in large language models, security vulnerabilities in generated code, architectural inconsistency, hidden operational costs associated with token consumption and licensing, and significant governance challenges.

This paper examines the current state of AI-assisted vibe coding, analyzes the technology hype cycle surrounding automated development, and evaluates the barriers to adoption within federal and defense environments. It also explores economic and political considerations related to workforce impacts, congressional oversight, and the redistribution of government spending toward private technology vendors.

The central conclusion is that AI-assisted coding will likely become an important productivity tool for developers, but it cannot replace disciplined engineering processes in large, secure, and long-lived government software systems. Further, when senior decision makers are faced with proposals that claim AI-assisted coding-related savings that feel "too good to be true" they probably don't factor all the costs associated with a shift of this magnitude. The proposed evaluation equation at the end of this paper will help those senior leaders factor all the variables to ensure a true "apples to apples" cost comparison when deciding on AI-assisted coding tradeoffs and anticipated savings.

Introduction

Large language models capable of generating executable code have introduced a new paradigm in software development. In this paradigm developers interact conversationally with AI systems that generate software components based on natural language prompts. This workflow is often referred to informally as "AI vibe coding." The developer describes the intent of a feature and the AI system produces a working implementation which is then refined and integrated into the broader system. In small development environments or early-stage startups this approach can significantly accelerate prototyping and early product development.

Federal enterprise software environments present a fundamentally different context. Government applications frequently operate within extremely large codebases developed over decades. These systems often contain millions of lines of code spanning multiple services, integration layers, and regulatory compliance frameworks. Many federal enterprise systems rely heavily on Microsoft or other enterprise server stacks. In Permuta's case, we have been writing code on top of the Microsoft stack for over 26 years. Our *Readiness*Platform family of solutions relies on this certified, accredited, and professionally supported infrastructure to provide assured secure, reliable and scalable solutions to the Department of War, Federal, State and Local Government and Higher Education sectors. The enterprise scale environments we operate in frequently include C# service layers, TypeScript web front ends, Microsoft SQL Server data tiers, on-premises Dynamics deployments, and hybrid integration with Azure Power Platform.

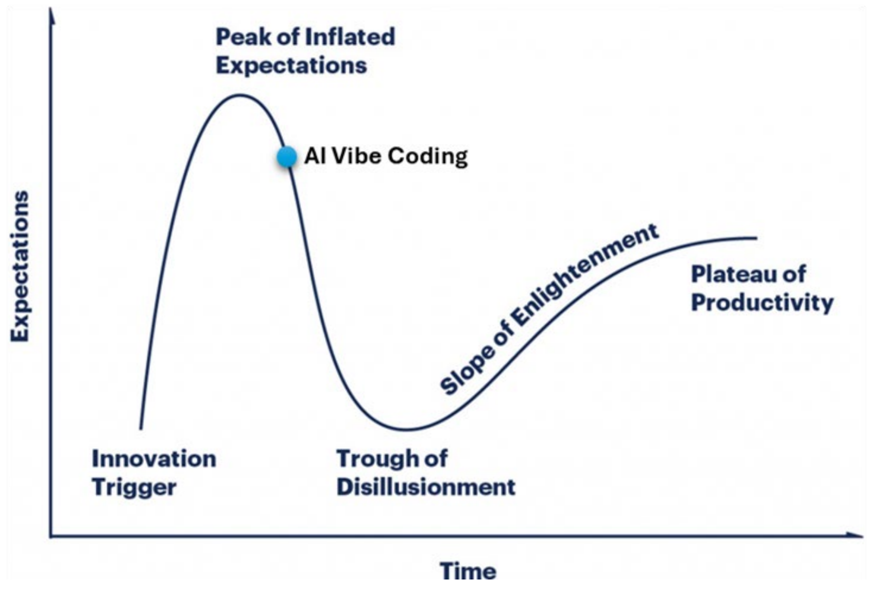
Such systems are not easily restructured or replaced. They represent long-lived infrastructure that must remain secure, maintainable, and operational for decades. In this environment the introduction of AI-generated development raises strategic questions for policymakers and technology leaders regarding security, architecture, workforce impact, and long-term program sustainability.

The AI Development Hype Cycle

The emergence of AI vibe coding tools closely follows the pattern described by the Gartner Hype Cycle. New technologies typically move through phases beginning with an innovation trigger followed by a rapid increase in public attention and expectations. This phase of inflated expectations is characterized by highly visible success stories and aggressive projections about the technology's future impact.

As organizations begin deploying the technology in complex real-world environments the limitations of the technology become clearer. This phase is commonly referred to as the trough of disillusionment. Over time practical use cases emerge and the technology eventually reaches a stable plateau of productivity.

Conceptual Gartner Hype Cycle with Estimated Position of AI Vibe Coding



AI vibe coding currently appears to be near the "peak of inflated expectations" heading downward toward the "trough of disillusionment." Public discourse frequently focuses on claims that AI systems will dramatically reduce software engineering staff or enable organizations to build complex enterprise systems with minimal human oversight. These narratives often overlook the realities of enterprise software development where architectural governance, security requirements, and long-term maintainability dominate engineering effort. Government systems represent an extreme example of this complexity.

Structural Constraints in Federal Software Systems

Enterprise government systems frequently contain millions of lines of code and incorporate multiple generations of architecture. These systems must integrate with legacy platforms, comply with regulatory frameworks, and maintain compatibility with operational environments that may persist for decades. Large language models operate with limited context windows that restrict the amount of source code they can analyze at one time. As a result, AI systems often lack the architectural awareness required to make consistent modifications across large codebases. When applied to monolithic systems, this limitation can lead to duplicated logic, inconsistent patterns, and architectural drift.

Security considerations introduce additional challenges. Government systems must comply with strict frameworks including the NIST Risk Management Framework, FedRAMP requirements, and Department of Defense secure development standards. AI-generated code has been shown in multiple studies to introduce security vulnerabilities at a higher rate than human-written code. These vulnerabilities frequently involve improper input validation, insecure authentication logic, and unsafe handling of sensitive data. Although these vulnerabilities can be mitigated through testing and review, they increase the burden placed on security teams and reduce the productivity benefits promised by automated development.

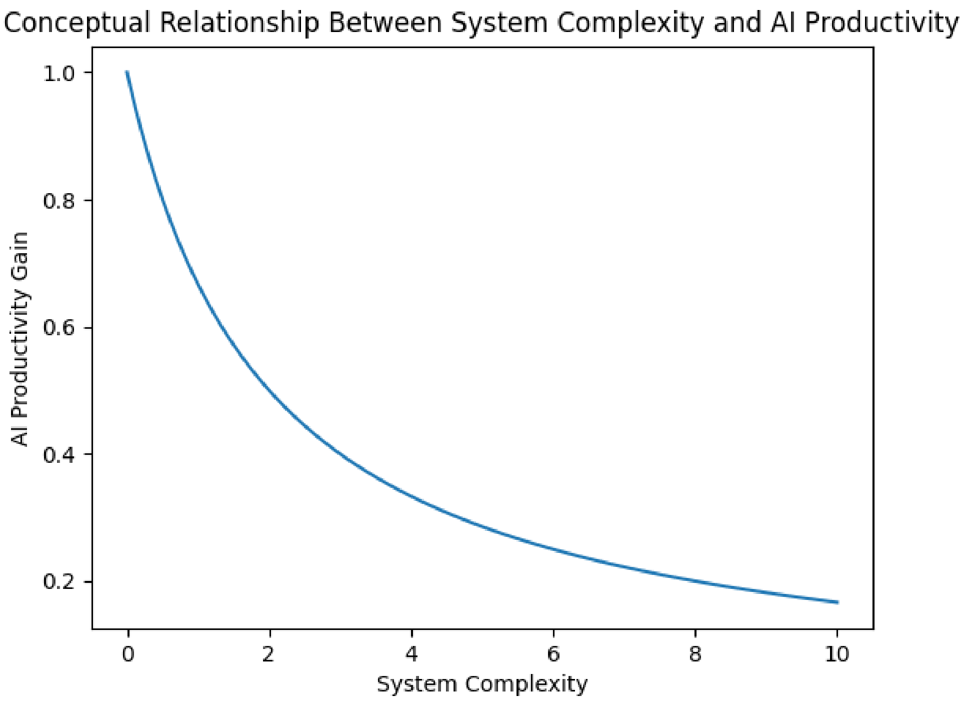
Architectural governance also becomes more difficult when code is generated dynamically through prompt-driven workflows. Enterprise systems rely on consistent architectural patterns to remain maintainable over long periods of time. When AI-generated components bypass these patterns the result can be increased technical debt and long-term maintenance costs.

Case Studies of AI Generated Code Failures

Several recent studies have demonstrated the security and reliability challenges associated with AI generated code. Research conducted by academic institutions and industry security groups has found that large language models frequently generate code containing common vulnerabilities including injection attacks, insecure authentication patterns, and improper data validation. Reports from the National Institute of Standards and Technology have emphasized the importance of rigorous testing and verification when using generative AI within software engineering environments. Defense organizations have similarly highlighted the importance of secure development practices within AI assisted workflows.

System Complexity and Productivity

Enterprise government systems often contain millions of lines of code and integrate with numerous legacy systems. As system complexity increases the ability of AI tools to reason about architecture decreases. Productivity improvements therefore tend to decline as system complexity grows. This relationship is illustrated by a conceptual curve showing how AI productivity gains diminish as system complexity increases.



According to Danial Braz, “AI does not simplify Architecture. It Exposes it…When AI is layered onto a weak architecture, it usually does not create new categories of dysfunction out of nowhere. It accelerates existing ones.” This is why the architectures associated with monolithic and complex enterprise systems cannot be shifted to AI orchestrated development and support in a wholesale manner without significant risk to the end customer’s data or mission. The reality is AI vibe coding just doesn’t scale in a linear fashion, especially in support of the Federal Government.

Hidden Operational Costs of AI Development

The discussion surrounding AI-assisted coding frequently focuses on potential labor savings. However, the operational cost structure associated with AI-generated development is often overlooked. Many AI development platforms operate on usage-based pricing models tied to token consumption and compute utilization. As organizations scale AI-assisted development these costs can increase substantially. Large monolithic systems exacerbate this problem because prompts must include larger volumes of context in order to produce meaningful code generation. This results in increased inference costs and higher token consumption.

Additional costs arise from the need to verify AI-generated code. AI generated development in these environments must navigate complex architectural boundaries including custom plugins, workflow automation, security roles, data governance policies, and legacy integration layers. These constraints significantly limit the ability of automated tools to modify systems without careful architectural oversight. Even when generated code appears correct it must still undergo security testing, compliance validation, integration testing, and code review. Testing costs may actually increase in some cases because developers must carefully validate code that they did not write themselves. Maintenance costs also become a significant factor if AI-generated code introduces inconsistencies in architecture or documentation.

The Subside Reality

Even considering the costs discussed above may not be enough to make a strategic decision. Many of the subscription and consumption costs behind the AI solutions and infrastructure required are currently propped up by heavy subsidies. As Claude Ciocan pointed out in a recent article, "For developers and founders building AI-powered products, this creates a dangerous foundation. Your unit economics might work beautifully today, but they're built on sand. When the subsidy era ends (and it will) the companies that survive will be the ones that saw it coming." According to the CEO of Perplexity, venture capital is propping up the layers of companies, solutions, and infrastructure or "wrappers" on top of the underlying power grid and data center compute costs that form the foundation of these AI tools. The venture capitalists are hoping their "wrapper" captures enough market share and becomes sticky enough to become profitable in the future. This venture capital-backed AI bubble has the potential pop resulting in significant price increases that could impact the viability of any business cases that involve wholesale replacement of human coders or significant shifts to reliance on these AI-assisted coding tools.

Institutional Barriers to Adoption

Technical and cost considerations represent only part of the challenge associated with AI-assisted development in government environments. Federal institutions operate within complex bureaucratic and political frameworks that influence technology adoption decisions. Government systems frequently support distributed workforces located across multiple states. Federal software programs often provide stable employment opportunities in regions that may have limited access to high-technology industries.

A shift toward AI-driven development that concentrates spending on a small number of technology vendors located primarily in Silicon Valley, Austin, and other AI-incubating cities could face substantial congressional scrutiny. Legislators are often reluctant to support policies that redirect federal spending away from their districts toward a concentrated group of technology companies.

Workforce considerations are particularly significant within the Department of War. Many military systems rely heavily on human expertise and operational judgment. In many cases the effectiveness of the weapon system is directly tied to the people operating and maintaining it. Proposals that focus exclusively on manpower reduction can therefore conflict with operational realities and institutional priorities. These political and organizational constraints mean that even if AI-assisted development offers productivity improvements, it may not translate directly into workforce reductions.

Any solution that's proposed savings are tied to reductions in military or civilian workforce should be treated with a great deal of suspicion. In my 30 years of service in the United States Air Force and the 5 years since, I had the opportunity to deliver enterprise software solutions at scale, run programs that implemented or addressed disruptive technologies. Without exception the single issue that resulted in a projects failure was projecting significant cost savings based purely on workforce reductions.

The Department of War is a collection of Services that derive their power in warfighting projection from a combination of military and civilian resources that comprise their wartime and peacetime workforces and in many cases weapon systems. The peacetime exercise of this weapon system through centralized control and decentralized execution requires a series of what often seems like inefficient communication or what many call bureaucracy. In reality it is a time-tested solution that minimizes complexity and training while maximizing effects and flexibility. Unfortunately, many look at this situation as inefficient and target it for automation or as an opportunity to save money without looking at the big picture.

Furthermore, Congress has a vested interest in maintaining the military and civilian work force in a distributed manner across the United States in financially isolated regions. This distributed network of military bases redistributes taxpayer dollars to areas that would otherwise be underrepresented. This combined with the above highlighted shift in AI-related revenue away from these congressional districts adds further congressional interest and risk to any AI-related disruptive technology that's viability hinges on workforce reductions alone and must be considered when making decisions about AI technology adoption and tradeoffs.

Toward a Quantitative Decision Framework

I am not saying that this disruptive technology has no place in today's software development world. Quite the opposite. I believe there are significant benefits to adopting these technologies to assist in development to drive innovation and productivity. I am just highlighting the fact that there is a significant amount of risk and variables to consider when making decisions about future expenditures tied to AI-assisted coding efforts.

Given the technical complexity and institutional realities described above, government organizations require a structured framework for evaluating the potential benefits of AI-assisted development. Most discussions of AI productivity focus on labor savings alone. This perspective fails to account for the numerous operational costs and organizational constraints that accompany the introduction of automated development tools. A more realistic approach is to evaluate AI-assisted development as a ratio between measurable benefits and the total cost and complexity introduced by the technology. The benefits of AI-assisted development generally fall into three categories.

1. The first category involves reductions in developer labor associated with code generation and debugging.
2. The second category involves improvements in development velocity that allow capabilities to be delivered more quickly.
3. The third category involves reductions in development cycle time that may accelerate program delivery.

Against these benefits must be weighed a broad set of costs that include software licensing, token consumption, testing overhead, maintenance complexity, architectural degradation, security risk, and governance requirements. In addition to these technical costs, government organizations must also consider political and institutional constraints related to workforce distribution and congressional oversight. Evaluating AI assisted development requires balancing potential productivity gains against operational costs and institutional constraints. A decision framework can be constructed that compares the expected benefits of automation with the combined technical, economic, and political costs associated with the technology.

Conclusion: The AI Development Decision Formula

The analysis presented in this paper leads to a decision framework that captures the balance between productivity gains and the full range of costs and barriers associated with AI-assisted development. The return on investment for AI-assisted development can be represented as the ratio between productivity benefits and the combined operational and institutional costs introduced by the technology. If that ratio results in a value of 1 or higher, then the savings outweigh the costs, and the effort should be considered viable based on a current understanding of all variables considered.

In simplified form the relationship can be expressed as:

$$AI_{ROI} = \frac{S_{labor} + S_{velocity} + S_{cycle}}{C_{license} + C_{tokens} + C_{testing} + C_{maint} + C_{architecture} + C_{security} + C_{governance} + C_{political}}$$

In practice this equation must also incorporate a complexity multiplier that reflects the scale and integration requirements of the system being analyzed. Large government codebases significantly increase the operational cost associated with AI-generated development due to architectural complexity and security requirements.

The expanded decision formula therefore becomes:

$$AI_{ROI} = \frac{(S_{labor} + S_{velocity} + S_{cycle}) \times P_{automation}}{(C_{license} + C_{tokens} + C_{testing} + C_{maint} + C_{architecture} + C_{security}) \times M_{complexity} + C_{governance} + C_{political}}$$

This framework highlights a key strategic insight. AI coding tools primarily function as productivity amplifiers for developers rather than replacements for engineering teams. In environments characterized by large monolithic systems, strict security requirements, and distributed workforces, the complexity multiplier and institutional constraints often offset the manpower savings predicted by AI vendors. For this reason, government and defense organizations should approach AI-assisted development as a capability that augments human expertise rather than replacing it. Organizations that adopt this balanced perspective will be best positioned to capture the benefits of AI-assisted development while preserving the reliability, security, and accountability required in mission-critical federal systems.

permuta

www.permuta.com

About the Author

General Manager, Product Group, Permuta Technologies
Martin “Marty” Jennings, Colonel (retired), United States Air Force

For the past 3 years, Marty Jennings has been the General Manager for the Product Group at Permuta Technologies, where he leads a team of Microsoft-certified developers responsible for more than 400 applications supporting unique Federal and Military use cases worldwide. Prior to joining Permuta, he served as Chief Cloud Architect and Chief Engineer for Leidos’ 10-year, $10B GSM-O II contract with the Defense Information Systems Agency (DISA). A 30-year U.S. Air Force veteran, Marty retired in the rank of Colonel in September 2021. His military and civilian career spans software development, cybersecurity, and large-scale enterprise system program management in support of multiple combatant commands and agencies.

LinkedIn: https://www.linkedin.com/in/jentek