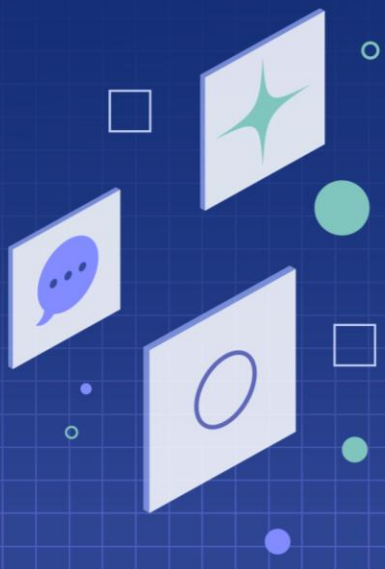
PEGA

WHITEPAPER

The token tax: Why autonomous AI agents are eroding the margins they were supposed to protect

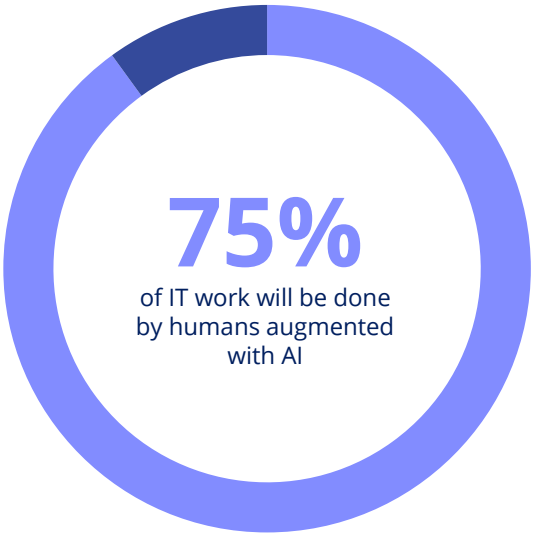
A guide for COOs, CXOs, and VP Operations Leaders



The opportunity is real

AI has changed software development as we know it. It is faster, more accessible, and more capable than it has ever been.

Developers can build and ship at speeds that seemed impossible even a year ago. 85% of developers are now using AI to accelerate their work (Stack Overflow Developer Survey, 2025). GitHub Copilot alone reports productivity gains of 51% faster coding speeds and 26% more completed tasks across its user base. Entire application prototypes that once took weeks are being scaffolded in days. The democratization of software development is real, and it is accelerating.



CIOs see where this is going. By 2030, Gartner expects that 75% of IT work will be done by humans augmented with AI, and 25% will be done by AI alone – with zero IT work untouched by AI (Gartner, November 2025). That is not a distant forecast. It is the planning horizon most enterprise IT organizations are operating against right now.

The investment is justified. The direction is right. But shipping more has not yet translated into the transformational business outcomes the board expects. And IT leaders are the ones being asked to explain why.

The gap is not a technology problem. It is an architecture problem. Three questions sit at the center of it, and most enterprises do not yet have clean answers to any of them.

1. ***Has AI development actually become more costly than simply hiring more engineers?***
2. ***Can AI-generated code be trusted to run enterprise operations at scale?***
3. ***How do you maintain visibility and control over what AI is building and deploying?***

The cost question: Are you paying more to build more debt?

The promise of AI-assisted development was clear: reduce the cost and time of building software. The reality, for many enterprises, is that the cost structure has shifted in ways that were not in the original business case.

Writing code is not the expensive part of software development. It accounts for roughly 10% of the total application lifecycle. The remaining 90% is review, testing, security patching, debugging, refactoring, and ongoing maintenance as business requirements change. AI code assistants have made the 10% faster. What the data is increasingly showing is that they have also made the remaining 90% more expensive.

GitClear's 2025 analysis of 211 million lines of code across repositories owned by Google, Microsoft, Meta, and enterprise organizations found an eightfold increase in duplicated code blocks in AI-assisted projects compared to two years prior. Code churn – lines revised or discarded within two weeks of being written – doubled from 2020 to 2024. As GitClear founder Bill Harding put it, AI-generated code functions like "a brand new credit card" for accumulating technical debt: the spending feels productive in the moment, and the bill arrives later. The State of Software Delivery 2025 report by Harness found that the majority of developers now spend more time debugging AI-generated code than building new features.

The token tax makes this worse. Every line of code an AI writes costs tokens to produce. Unlike a token spent on a customer interaction, a token spent generating application code produces an asset that requires ongoing investment to maintain. If that asset accumulates debt faster than it delivers value, the economics compound in the wrong direction from day one.

And the cost structure of AI development tooling itself is no longer predictable. GitHub Copilot moved to consumption-based billing in June 2026, replacing its flat-rate premium request unit model with token-based AI credits. GitHub's own explanation acknowledged that "a quick chat question and a multi-hour autonomous coding session can cost the user the same amount" under the old model, and that absorbing escalating inference costs "is no longer sustainable" (GitHub Blog, April 2026). Enterprises that budgeted against flat-rate AI development costs are now operating in a variable-cost environment with less visibility into what they are spending per workflow.

The real question is not whether AI development tools are worth using. They are. The question is whether the current approach – using AI to generate more code faster – is actually reducing total development cost, or simply front-loading delivery speed while compounding the maintenance liability that shows up in next year's budget.

The performance and security question: Can you trust what AI builds?

Faster delivery only creates value if what gets delivered can be trusted to run at enterprise scale. On this dimension, the evidence is giving IT leaders legitimate reason for caution.

CodeRabbit's analysis of 470 real-world pull requests found that AI-generated code introduces 1.7 times more total issues than human-written code, and 2.74 times more security vulnerabilities specifically (CodeRabbit State of AI vs. Human Code Generation, December 2025). IEEE research presented at ISSRE 2025 found that AI-generated code exhibits higher rates of high-risk security vulnerabilities despite producing structurally simpler outputs – suggesting the risk is in how the logic is constructed, not just its complexity.

AI tools generate solutions to immediate, local problems. They do not have full awareness of your system's broader architecture, your internal abstractions, your shared utilities, or the security constraints that govern how data moves through your environment. The result is code that passes unit tests and clears initial review, but introduces vulnerabilities that only surface in edge cases or under production load.

76% of developers using AI coding tools report generating code they did not fully understand at least some of the time (Stack Overflow, 2026). That statistic carries real operational weight. When a CIO is asked by a regulator, a legal team, or a board why a specific system behavior occurred, "the AI generated it and the developer wasn't sure how it worked" is not an acceptable answer. And yet that is the implicit accountability posture of any organization deploying AI-generated code into production without a structural governance layer.

Gartner's 2026 top predictions include a forecast that "death by AI" legal claims will exceed 2,000 globally, driven by insufficient governance over AI-generated application outputs (Gartner, October 2025). These are not hypothetical future risks. They are active exposure for enterprises that have given AI-generated code meaningful authority over regulated or customer-facing workflows today.

The reliability question: Do you know what AI is building?

Speed and security are operational concerns. Auditability is a structural one – and for CIOs and enterprise architects, it may be the hardest to retrofit after the fact.

When an application makes a decision, whether routing a case, determining eligibility, granting access, or generating a customer-facing output, IT is accountable for that decision – not the AI that wrote the code. The IT organization deployed the application and the leaders approved it.

In a traditional development environment, that accountability is manageable because the logic is inspectable. When an auditor asks why a decision was made, a developer traces the relevant code path and finds the answer. In an environment where application logic was generated by AI, that traceability is not guaranteed. AI-generated code produces working output, but the reasoning behind specific implementation choices is not documented in the way that intentional human engineering is. When the audit question is "why did this system make this decision in this edge case," the answer "because the AI wrote it that way" is not a defensible response.

This is the reliability problem that matters most at the enterprise level. It is not about whether the code runs correctly on average. It is about whether you can explain any specific outcome to anyone who asks, at any point in the future, in language they can act on.

Visual, declarative application logic solves this problem structurally. When business rules, decision tables, and workflow routing are expressed as inspectable configuration rather than generated code, every decision path is traceable without forensic analysis. The logic exists at a layer that compliance teams, legal teams, and business stakeholders can read, not just engineers. Audit becomes a configuration review, not a code archaeology exercise.

The architecture that answers all three

The architectural shift that resolves cost, security, and reliability at once is not a constraint on AI use. It is a redirection of where in the development stack AI is applied.

AI is genuinely effective at accelerating the generation of high-level application structure: data models, workflow definitions, decision tables, routing logic expressed in declarative form. These are artifacts that describe what an application does without encoding implementation in imperative code. They are readable, auditable, and maintainable in a way that generated code is not. When business requirements change, the model changes, and application behavior changes with it, without the cascading code modifications and regression risk those changes introduce.

This is the model-first development approach in practice. AI does the work it is actually good at: translating intent into structure, generating configuration from natural language, accelerating architectural definition. The output is a model, not code. Code generation, where necessary, happens underneath it as an implementation detail, not as the primary artifact.

The operational implications are concrete. Developer teams spend less time on maintenance because the maintenance surface is the model, not the codebase. Security review is more tractable because business logic is expressed declaratively. Compliance audits are faster because the decision layer is inspectable without engineering support. And token spend is concentrated on the highest-value part of development: defining what the application should do, not generating the mechanics of how it does it.

Only 28% of AI infrastructure use cases fully meet ROI expectations, according to Gartner's I&O survey from late 2025. The gap between expectation and reality is not a capability failure. It is an architecture failure. Enterprises capturing ROI from AI in development are not necessarily using more advanced models. They are using AI to build artifacts that do not depreciate at the rate that generated code does.

The architecture decision

The choice IT leaders must make is not between speed and safety. That framing is what makes the current situation feel like an impossible constraint. The actual choice is between two different architectures with very different long-term cost and risk profiles.

An architecture centered on AI-generated code delivers fast initial output with a compounding maintenance tail, limited auditability, and security exposure that grows with the size of the codebase. An architecture centered on model-first development with embedded AI delivers initial output that is auditable by design, maintainable at the model layer, and generates token spend on the part of development that actually creates durable value.

IT leadership does not have to say no to the business's demand for AI-accelerated delivery. The answer is not slower. The answer is a different development philosophy that the business can sustain and IT can defend.

The right AI investment is not writing code faster. It is building application models that do not need to be rewritten.

Thank you

About Pega

Pega provides the leading AI-powered platform for enterprise transformation. The world's most influential organizations trust our technology to reimagine how work gets done by automating workflows, personalizing customer experiences, and modernizing legacy systems. Since 1983, our scalable, flexible architecture has fueled continuous innovation, helping clients accelerate their path to the autonomous enterprise. **PEGA.COM**