SECURE CODE WARRIOR

From code writer to creator & orchestrator:

The SCW AI Adoption Model™

Authors
Pieter Danhieux, CEO & Co-Founder, Secure Code Warrior
Dr. Matias Madou, Ph.D., CTO & Co-Founder, Secure Code Warrior

SECURE CODE WARRIOR

Table of contents

Introduction 1

AI adoption naturally introduces risk 2

The governance gap is plaguing enterprises 3

The road to agentic engineering 4

Phase 1: Minimal AI use 4

Phase 2: Supervised assistance 5

Phase 3: Unsupervised assistance 6

Phase 4: AI-primary development 7

Phase 5: CLI single agent 8

Phase 6: Multi-agent parallel 9

Phase 7: Scaled agent management 10

Phase 8: Autonomous orchestration 11

Further reading on risk inflections: the two stages where risk levels spike 12

The cost crisis: why CISOs need to act now 13

Conclusion 14

Introduction

Agentic AI software development systems are outrunning traditional approaches to governance, exposing enterprises to mounting risks, ballooning technical debt and increasing costs. To close that gap, Secure Code Warrior ("SCW") has implemented the SCW AI Adoption Model™ outlining 3 phases and 8 stages of AI adoption for software developers, from those who minimally use AI to fully agentic engineering. An approach that will accelerate AI adoption, with the developer evolving from code writer to orchestrator and creator, with the security skills required at each stage.

Generative and agentic AI are transforming software development, dramatically increasing the amount of code written with mixed quality/security results while shifting developers from writing code to reviewing output and ultimately overseeing it. But, most organizations lack the governance to match. As AI adoption accelerates, the risks change, costs become harder to manage, and there is a general loss of visibility across the development lifecycle on who or what created the code.

For all of the productivity gains it delivers, AI-generated code isn't reliably secure. SonarSource's Leaderboard for Code Quality and Security makes for sobering reading: the data currently indicates that the models most capable of writing code that runs successfully often produce code with more severe security vulnerabilities. Even the very best models introduce between **18 and 26 general issues per 1,000 lines** of code (KLOC). Maintainability issues and flaw density remain too high for unsupervised use, especially in the enterprise.

Our proprietary benchmarking research across **660** AI-generated codebases also found that AI systematically introduces specific flaws across every model and framework tested, including overly permissive defaults in **87%** of codebases, sensitive data in logs in **58%**, and hard-coded credentials in **52%**. Even our proprietary research found significant differences in security performance across AI models, making model choice an ongoing factor in risk aversion.

AI code is increasingly embedded across enterprises and is being experimented with outside teams with technical expertise, giving untrained "citizen developer" employees unchecked access. This creates even more potential for poor security outcomes, expanding the attack surface and making organizations vulnerable to attacks that are, not surprisingly, increasingly fueled by AI tools used by malicious actors. Every board is questioning the CEO about an "AI strategy".

Every CEO sees AI as a potential threat to their business, while also recognizing the opportunity to move faster and more efficiently. In the race to remain relevant, they are driving AI adoption across their organizations and giving teams a "carte blanche" directive to experiment with AI tools and data, sometimes in direct defiance of their CISOs' advice.

A significant finding in CrowdStrike's 2026 Global Threat Report is that AI-enabled attacks, which increased by **89%** in 2025 over the previous year, are targeting AI systems themselves. AI systems embedded in development pipelines (along with workflows, SaaS applications and just about everywhere else) have become prime targets.

The weakness that prevents organizations from defending against this growing threat stems from the fact that many of them are working in the dark when it comes to securing AI implementations. Despite the breakneck speed at which enterprises are adopting AI, most lack visibility into how AI is used, who uses it, and how it contributes to production code. Organizations are operating without essential protective measures, such as oversight, guardrails to ensure AI is used safely and responsibly, and developer-level observability. The fact that these tools are so often used by employees who lack the skill sets to manage complex security assessments and perform contextual reviews also poses a significant threat to the integrity of the security program and the overall organizational risk level.

Enterprises can no longer ignore this risk. The expansion of the attack surface created by unrestrained and unauthorized AI use (often under the radar of security teams and business leaders) and the accompanying threat from malicious actors pose clear risks, especially as AI tools become more ubiquitous and autonomous. Gartner projects that through 2027, the costs from task-driven AI agent abuses will increase fourfold; and, Gartner's 2026 Hype Cycle for Secure Software Engineering notes that AI-augmented development is 'expanding the attack surface faster than traditional controls can scale'. Organizations need a governance plan that fully addresses the new reality of software development.

To that end, Secure Code Warrior has implemented a three-phase, eight-stage framework that reflects the clear progression in AI adoption, from minimal AI-assisted tasks to agentic autonomy, and addresses the rising risk levels associated with each stage. The model will help CISOs connect AI usage by employees, development visibility, code contributor capability, commit activity, and software risk signals, enabling them to measure, reduce, and govern risk while demonstrating progress in securing the Software Development Lifecycle (SDLC) as it transitions to the more relevant Agentic Development Lifecycle (ADLC).

AI adoption naturally introduces risk

The proliferation of AI across enterprises has developed a life of its own and brought risks with it as part of the package.

Attack surfaces are spreading right under the noses of IT and security teams, through shadow AI and unauthorized AI automations. The push to boost productivity and reduce costs is putting teams under pressure to perform, so many employees don't ask permission to make use of GenAI and AI agents, according to a recent Gartner report. They readily embed AI automation to generate reports, use vibe coding or low-code/no-code platforms to customize applications, and integrate third-party resources on their own. And they do this all out of sight of supervision. In fact, **32%** of IT workers using GenAI tools actively try to hide them from cybersecurity teams, Gartner reports.

32% IT workers using GenAI tools actively trying to hide them from cybersecurity teams

Shadow IT and off-the-cuff automations aren't the only causes of software weaknesses, but they have contributed heavily to an environment rife with vulnerabilities that put organizations at risk from damaging cyberattacks and regulatory violations. Gartner warns that 'agentic coding patterns like vibe coding enable software development without formal methods such as design creation, code reviewing, or security verification — and vibe-coded applications will work their way to production without proper security verification[1]. Cybercriminals and other malicious actors, for their part, are happy to exploit the security gaps created by AI use. And by using their own AI tools, they are carrying out their attacks more quickly, with an average breakout time — the interval between initial access and lateral movement in the network — of 29 minutes, a **65%** increase in speed from the previous year.

It's also notable that **82%** of detected attacks did not contain malware, as attackers increasingly focus on compromising trusted identities, SaaS integrations and inherited supply chain elements, CrowdStrike said. Gartner likewise projects that through 2029, more than half of successful cybersecurity attacks against AI agents will result from attackers exploiting access control issues while using direct or indirect prompt injection as an attack vector.

Secure Code Warrior's proprietary benchmarking research discovered that the dominant vulnerability (found in **40%** of generated codebases) is Protection Mechanism Failure (CWE-693). Why is this significant? Essentially, this reveals that the risk profile is determined not only by the impact of poorly written code but also by the AI's complete omission of security best practices like authentication because it was never prompted to include them.

Without skill, the AI will write perfect code for the wrong problem.

[1]Gartner Hype Cycle for Secure Software Engineering (2026). Harrison, A.

The governance gap is plaguing enterprises

Fair or not, much of the responsibility for addressing the problem falls on developers. Software developers have a kind of love-hate relationship with AI tools, in that they can't live (and keep up with production demands) without them, but sometimes seem as though they'd rather not have to live with them.

Stack Overflow's 2025 Developer Survey found that **84%** of respondents are using or planning to use AI tools, up from **76%** the previous year; Stack Overflow expects that **51%** of developers will use these tools every day. However, using the tools doesn't mean they have faith in them. Only **33%** trust their AI tools (and only **3%** say they highly trust them), while **46%** actively distrust AI-generated code. And the more experienced developers were, the less likely they were to trust AI, the survey report notes. Nevertheless, they still use AI tools, even if they have doubts about them.

84% Developers using or planning to use AI tools

33% Developers who trust their AI tools

This may put developers in something of a bind, but the biggest problem isn't with developers or even AI. It's a lack of governance and the fact that AI scales beyond an organization's ability to control it.

A lack of governance amplifies three risks:

1

Unattributed risk. With productivity as the primary driver in the CI/CD pipeline, AI-generated code can be pushed into production without a clear, discernible origin. This can open a company up to audit and compliance exposure, an ambiguous chain of accountability and slower incident response.

2

Unbounded cost. When AI use scales without constraint, the costs can add up quickly and leave organizations trying to budget for unpredictable spending, reduced ROI and inefficient usage of AI models. Gartner predicts that by 2027, more than **40%** of agentic AI projects will be abandoned because of uncontrolled costs, undetermined value and poor risk controls.

3

Uncontrolled velocity. The speed of AI-assisted development can easily exceed humans' capacity to review and validate the software, creating critical bottlenecks. That scenario can lead to a breakdown of the SDLC, increased technical debt and slower delivery of software despite higher output.

At the root of the problem is a lack of visibility. Despite their rapid adoption of AI, most organizations lack visibility into exactly how AI contributes to production code. Existing security programs designed to monitor and manage human-authored code are not equipped for the AI-generated code era. Security leaders cannot answer the questions that matter:

- Which AI models contributed to specific commits?
- Does AI-generated code meet secure coding standards?
- Which developers introduce elevated software risk?
- Are development practices improving over time?
- Is AI development going through a robust software development lifecycle?

Without the ability to closely monitor AI-related activity, from developer education to commits made by AI to the provenance of production code, an organization runs the risk of losing control of software quality and safety, as AI agents continually accelerate production without security checks. That's where a comprehensive governance plan comes in.

The road to agentic engineering

Not all AI systems are the same. Organizations may talk about AI coding as if it were one thing, but AI-assisted software development evolves as the importance and impact of its role within the enterprise grow. It follows a predictable maturity curve across eight observable stages, which can be grouped into three phases. An adoption model that mirrors those stages can enable organizations to maintain visibility and enact governance throughout the lifecycle.

This **AI Adoption Model** maps each stage of AI-assisted development to the right combination of developer upskilling and governance controls, showing how SCW Learning builds the capabilities developers need at each inflection point and how SCW Trust Agent maintains visibility and enforces policy as autonomy increases.

PHASE: AI-ASSISTED

1 Minimal AI use

Risk level: low

At this phase, developers are just getting started with AI. AI supports development, but humans remain the primary authors. This is the ideal time to build a governance foundation before oversight degrades, with Trust Agent establishing the baseline.

Organizational indicators: At this foundational stage, an organization's AI footprint is minimal. Developers rely on public generative AI tools like ChatGPT, DeepSeek-R1, or Mistral, primarily for quick lookups and debugging, treating them as an advanced, quick-fire alternative to traditional forums like Stack Overflow. Formal AI coding tools are not integrated into the development workflow or IDE, and organizational policies surrounding AI assistance are typically restrictive or non-existent. As developers continue to manually author the software, the organization's overarching security practices remain largely unchanged from the pre-AI era.

Developer experience: The daily engineering workflow feels highly traditional. Developers write every line of code themselves, conduct human-to-human code reviews, and intentionally ignore or disable AI features within their IDEs. While an engineer might occasionally paste an error message into an AI chat, the output is treated strictly as unverified reference material that must be carefully scrutinized before use.

Security posture and practices: Due to developers retaining full authorship, traditional security paradigms apply perfectly: if you write it, you own it. Security accountability is absolutely clear, and vulnerability knowledge is directly actionable because the developer intimately understands every line of code. Established security gates, such as SAST, DAST, and penetration testing, continue to operate exactly as they did before the introduction of AI.

Key learning concepts: Even with minimal AI adoption, teams must update their awareness. Developers need to recognize that AI-generated answers can be subtly misleading or outdated, may hallucinate, and serve only as a starting point rather than a definitive source of truth. It is vital to establish a clear baseline of what "secure code" looks like in your specific framework, mandate the verification of all code regardless of its origin, and train developers on the severe risk of context leakage when pasting proprietary data into public tools.

Primary security responsibility: The core mandate remains unchanged: write secure code from the very beginning and actively apply secure coding patterns in every implementation.

Key risks: The most critical vulnerability at this stage is blind trust, specifically, the danger of developers copying and pasting AI-generated snippets from external chats without conducting a rigorous security review.

SCW training will focus on foundational AI security literacy, Common Weakness Enumeration (CWE) awareness, and a secure coding baseline. Trust Agent will set a Trust Score baseline and establish behavior tracking.

PHASE: AI-ASSISTED

2 Supervised assistance

Risk level: moderate

Organizational indicators: At this stage, formal AI coding assistants, such as GitHub Copilot or Cursor, are officially integrated into the IDE, typically guided by an organization's acceptable use policy. While AI actively generates code suggestions, the dynamic remains strictly supervised: an explicit permission model requires developers to review and approve every single AI action, ensuring they retain full control over the final output.

Developer experience: The daily workflow shifts toward acceleration rather than replacement. AI suggests completions as you type and assists via IDE chat to generate boilerplate or propose refactoring. Despite the noticeable productivity gains on repetitive tasks, you still write the core logic and feel like the true author of the code, conducting line-by-line reviews for every pull request you submit.

Security posture and practices: Security at this stage hinges on the developer's ability to actively review AI-generated code before acceptance. A new "code reviewer" role emerges, requiring developers to scrutinize AI output with the same rigor they apply to human colleagues, while relying on SAST tools to catch any misses.

Crucially, traditional vulnerability knowledge must be applied to spot AI's predictable blind spots.

Key learning concepts: To operate safely, developers must learn the specific vulnerability patterns that consistently appear in AI-generated code. This requires reading suggestions critically for unhandled edge cases, applying the OWASP Top 10 as a mental checklist, and mastering secure prompt engineering, such as explicitly commanding the AI to "use parameterized queries" or "require authentication". Teams must also thoroughly understand the permission models of their IDE integrations to know exactly what the tool can access.

Primary security responsibility: The fundamental duty is to rigorously review AI-generated code for security flaws before acceptance, maintaining the same security standards required for human-authored code.

Key risks: The greatest danger at this stage is habit-forming acceptance. As AI suggestions are often right, developers can fall into a trap of reflexively approving them without deep review. The resulting flaws are often subtle and architectural rather than syntactic, meaning if you don't explicitly tell the AI tool to add authentication, it simply won't.

The greatest danger at this stage is habit-forming acceptance

AI security training is mapped to supervised workflows, code review awareness, and Trust Score tracking. Trust Agent will perform traffic interception between the integrated development environment (IDE) and large language model (LLM), as well as LLM identity tracking and acceptance monitoring.

PHASE: AI-ASSISTED

3 Unsupervised assistance

Risk level: high

! RISK INFLECTION

Organizational indicators: At this stage, organizations experience a rapid surge in development velocity, often outpacing their existing governance processes. A paradigm of "vibe coding" emerges, in which developers simply describe their desired outcomes and grant AI tools broad permissions to generate significant portions of code without explicit per-action approval. As a direct result of this unsupervised generation, invisible security debt begins to systematically accumulate across the enterprise.

Developer experience: The daily workflow fundamentally shifts from intimate, line-by-line authoring to high-level orchestration. You provide broad instructions to the AI and accept most of its solutions without deep scrutiny, simply because they "usually work". While your perceived productivity is dramatically higher, it comes at the cost of codebase familiarity; you may find yourself submitting pull requests containing code you do not fully understand.

Security posture and practices: This marks Risk Inflection Point #1: vulnerabilities are now being introduced into code that developers have never read in depth. With human oversight diminishing, automated SAST and SCA pipelines become critical compensating controls. However, our research demonstrates that a single SAST tool is no longer sufficient; achieving adequate coverage requires multiple independent scanners to detect AI-generated flaws. Framework selection also elevates to a primary security decision, as frameworks with opinionated security defaults reduce risk by an order of magnitude. Additionally, because the AI autonomously pulls in dependencies, supply chain risk increases, though the surface of vulnerabilities remains highly predictable: our research identifies **87** unique CWE categories across AI code, with **39** produced by every evaluated model on average.

87 Unique CWE categories across

39 CWE categories produced by every evaluated model on average

Key learning concepts: Development teams must urgently address the "trust trap", or, the dangerous habit of suspending critical review just because a tool is usually correct. Developers must be trained in "blast radius" thinking and the risks of excessive agency, recognizing that granting broad permissions to an AI is fundamentally different from trusting a human who understands intent. They must also understand indirect prompt injection and the inherent risks of vibe coding when they cannot explain the underlying code.

Primary security responsibility: This is where the developer's core mandate evolves. You must maintain acute security awareness even when you are no longer reviewing every line of code, and you must know exactly when to escalate a change for deeper review based on risk classification frameworks.

Key risks: The paramount risk is the accumulation of invisible security debt driven by over-trust and systemic oversight failure. This debt is a proven reality: research confirms that **87%** of AI-generated codebases contain incorrect permission assignments, and **52%** contain hard-coded credentials.

Training covers AI observability, advisory policies and adaptive learning triggers based on over-trust patterns. Trust Agent performs over-trust detection and MCP service tracking, while implementing an adaptive learning trigger and a vulnerability-based training pathway.

This is where the concept of "invisible security debt" meets reality: it accumulates largely because, at this point, developers stop reading the code deeply.

PHASE: AI-NATIVE

4 AI-primary development

Risk level: high

When AI adoption reaches this point, developers trust AI output without rigorous review. Oversight is already deteriorating if not addressed by enforcing a governance policy. Trust Agent detects this and automatically triggers the appropriate training.

Organizational indicators: At this stage, the AI agent takes over as the primary code generator, and developers effectively stop writing code themselves. Interaction shifts entirely to reviewing diffs and providing high-level direction. While overarching architectural decisions remain developer-led, the AI proposes and produces the actual implementations. As a result, traditional code review processes must rapidly adapt to handle the unprecedented volume of AI-generated output.

Developer experience: Your daily workflow transforms from hands-on coding to architectural orchestration. You describe the desired outcome, and the AI handles the granular implementation details. The role feels less like an individual contributor and more like a tech lead reviewing a junior developer's work. However, this elevated vantage point comes with a severe tradeoff: you begin to lose the ability to mentally model exactly what the application's code does at a deep level.

Security posture and practices: Security knowledge fundamentally shifts from "how to write secure code" to "how to spot insecure patterns". Architecture-level security and formal threat modeling become far more critical than line-level review, necessitating comprehensive automated security testing to keep pace. LLM-specific vulnerabilities, such as prompt injection and excessive agency, become active, primary threats.

87% AI-generated codebases containing incorrect permission assignments

52% AI-generated codebases containing hard-coded credentials

Our research demonstrates that model performance is non-transitive across frameworks, meaning a model that writes secure code in one framework may rank last in another. This discovery determined that model selection must be evaluated on a per-stack basis, not organization-wide.

Key learning concepts: Developers must master security-focused diff review; that is, the distinct skill of evaluating code without having written the surrounding context. Teams must also learn to threat model at the design level, implement secure LLM design patterns (such as dual-LLM or the "Rule of Two"), and critically evaluate the trust assumptions of AI communication protocols like MCP, Skills (by Anthropic), and agent-to-agent (A2A). It is also critical that strict "provenance" is established and maintained: every line of code must be traceable back to a human decision, reinforcing that the developer reviewing the commit ultimately owns the security of the output.

IMPORTANT: Model Context Protocol (MCP) and "Skills" awareness must be prioritized at this stage. As AI agents integrate more deeply, they call upon external tools either via MCP or through Anthropic's newly introduced Skills architecture. Because Skills rely on natural language, they are vulnerable to instruction-based weaknesses and vulnerabilities. Teams must rigorously monitor which external services their AI connects to, what data flows through those connections, and actively utilize tools like NVIDIA's open-source scanners to identify and remediate weaknesses in Skill instructions before deployment.

Primary security responsibility: Your core mandate is to ensure that AI-generated architectures and implementations strictly meet security requirements through rigorous design review, threat modeling, and comprehensive automated testing.

Key risks: The greatest danger is the developer's loss of deep code comprehension, which allows critical architecture-level flaws to slip past superficial line-level reviews. Developers must explicitly supply the architectural intent and security parameters that the AI simply cannot infer.

SCW provides code review training for reviewers, but not authors, along with commit-level risk scoring and MCP policy gating. Trust Agent monitors the percentage of LLM-generated code, maps developer identities, and then creates a Software Bill of Materials (SBOM) and a governance model.

PHASE: AI-NATIVE

5 CLI single agent

Risk level: very high

Organizational indicators: At this stage, development moves beyond the IDE. Developers rely on CLI agents, such as Claude Code or Gemini CLI, for end-to-end task execution, with a single agent processing entire features from description to full implementation. Owing to the fact that code diffs scroll past rapidly, per-line human review is effectively eliminated. The developer's role shifts almost entirely to prompt engineering and constraint-setting, making version control and CI/CD pipelines the organization's primary safety nets.

Developer experience: Your daily work moves to the terminal, where you orchestrate rather than edit. You describe features in natural language, and the agent produces complete implementations. When the sheer volume of generated code far exceeds what any individual can manually review in detail, your focus shifts to setting architectural constraints via rules files and scanning high-level diffs at the pull request stage.

Security posture and practices: With manual code review capacity vastly exceeded, automated security tooling and the CI/CD pipeline become the definitive quality gates. Security "rules files" effectively evolve into your formal security policy. The introduction of autonomous agents elevates Non-Human Identity (NHI) management to a primary security concern; agents require credentials for APIs, repositories, and environments. In addition, the external tools and MCP servers that these agents connect to introduce significant new compromise risks.

Key learning concepts: Developers must urgently master a new operational security layer. This requires applying the "minimal footprint" principle, ensuring that agents request only task-specific permissions rather than broad standing access. Teams must rigorously manage NHI credentials through scoping, rotating, and auditing, and learn to write secure system prompts that tightly constrain agent behavior. Additionally, awareness of novel attack vectors is critical: developers must guard against prompt injection via the CLI (from files, environment variables, or APIs), secure the CI/CD pipeline against injection attacks, and understand how compromised external MCP servers can maliciously instruct an agent to exfiltrate data or execute destructive actions.

Primary security responsibility: Your core mandate is to define and enforce strict security constraints through agent configuration, automated tooling, and rigorous CI/CD gates, while securely managing all agent credentials.

Key risks: The most severe risks at this stage are the sheer volume of code exceeding human review capacity, the mismanagement of powerful agent credentials, and the threat of MCP-driven prompt injection originating from malicious external data sources

At this stage, a Command-Line Interface (CLI) operates autonomously. Users receive training on enforced policies, automated risk scoring, MCP governance and prompt injection defense. Trust Agent performs CLI traffic interception, tand provides an AI system of record, as well as MCP provenance tracking.

PHASE: AGENTIC

6 Multi-agent parallel

Risk level: critical

! RISK INFLECTION

At this phase, you're moving faster than your security program can keep up. Policy-driven governance is no longer something you can leave in the "too hard" basket.

Organizational indicators: At this advanced stage, organizations deploy three to five specialized agents working simultaneously across different components, such as front-end, back-end, and testing. Individual code review becomes physically impossible at this extraordinary velocity; the organization must rapidly shift from manual review to policy-based governance. Developers transition into mediators who carry context across these agents, while A2A communication protocols become a core part of the infrastructure.

Developer experience: Your daily work transforms into orchestrating multiple agents and resolving conflicts between them. Instead of authoring or reviewing code, you act primarily as a context broker and architect. While development speed is astonishing, your oversight is strictly limited to high-level checks; as you cannot realistically review all the code being produced, **you must place your full trust in automated security gates.**

Security posture and practices: This marks Risk Inflection Point #2, where the volume of output makes human review physically impossible, leaving automated security gates as your only line of defense. The primary attack surface shifts toward agent communication security (MCP, A2A), where naming attacks, shadowing, and context poisoning emerge as active threats. Additionally, supply chain risks multiply exponentially as each autonomous agent pulls different dependencies, making strict agent authorization boundaries and least-privilege permissions absolutely critical.

Key learning concepts: To survive at this velocity, teams must master "policy-as-guardrail," defining and enforcing rules such as rate limits, tool restrictions, and data access scopes without per-action approvals. Developers must understand the risks of agent coordination and explicitly treat output from one agent to another as untrusted input, just like any external data source. Teams must also learn to secure A2A protocols against rogue instructions, manage the expanded supply chain for agents (including base images and MCP servers), maintain traceability when multiple agents commit simultaneously, and apply risk-based triage to prioritize human inspection of high-risk outputs, such as authentication changes.

To survive at this velocity, teams must master "policy-as-guardrail," defining and enforcing rules such as rate limits, tool restrictions, and data access scopes without per-action approvals

Primary security responsibility: Your fundamental duty is to define and enforce strict agent boundaries, authorization policies, and automated security gates, while continuously monitoring for cross-agent compromise.

Key risks: The most severe vulnerabilities arise from the physical impossibility of human review,cross-agent contamination via protocol attacks, and cascading compromise that spreads through shared context.

Having multiple agents running in parallel allows AI to scale up quickly. The training covers the policy enforcement engine, application risk scorecards, and commit traceability for audit purposes. Trust Agent provides container support, LLM performance monitoring, and cross-agent traceability.

Further identified risks such as "cross-agent contamination" and "context poisoning," necessitate that automated gates become the only line of defense.

PHASE: AGENTIC

7 Scaled agent management

Risk level: extreme

Organizational indicators: This is the frontier stage, where organizations push the absolute limits of manual orchestration, with developers hand-managing **10 or more** agents operating concurrently. These agents share context and continuously coordinate across complex tasks. Because human oversight can no longer scale, security governance must be fundamentally systematized: rigid, automated policies (rather than individual developers) now enforce security, necessitating comprehensive and immutable audit trails across all agent activities.

Developer experience: Your daily workflow shifts to high-level system management. While you set the overarching architectural direction, you can barely track what each individual agent is doing. The sheer volume of orchestrating **10+** agents is overwhelming, forcing you to rely entirely on dashboards, logs, and alerts rather than direct code review. When agent failures or conflicts occur, you are forced to perform rapid triage without full context, leaving you with the distinct feeling that the system is running ahead of your ability to oversee it.

Security posture and practices: Security at this scale requires robust, formal frameworks for agent authorization that dictate exactly what each agent can and cannot do. Separation of duties between agents elevates to a primary security architecture concern. With human review rendered impossible, organizations must rely on comprehensive automated checks to manage the complete agent lifecycle, from onboarding to decommissioning. Mandatory audit trails must capture exactly which agent did what, when, and why, while automated risk scoring at the commit level is used to triage the rare changes that still require human attention.

Security at this scale requires robust, formal frameworks for agent authorization that dictate exactly what each agent can and cannot do.

Key learning concepts: Development teams must elevate their skills from component-level to system-level analysis. This requires mastering system-level Architecture Risk Analysis and formal threat modeling (such as STRIDE or PASTA) to design the constraints fed to agents. With 10+ agents operating under distinct credentials, Non-Human Identity (NHI) lifecycle management becomes a major security domain, requiring precise agent-authorization frameworks and strict least-privilege enforcement. Teams must also implement “policy-as-code” to evaluate machine-readable rules at every commit, use predictive risk models to detect emerging AI vulnerability patterns, and maintain acute regulatory awareness to ensure that autonomous generation complies with mandates such as PCI DSS, GDPR, and the CRA.

Primary security responsibility: Your central mandate is to architect and meticulously maintain the security governance framework itself, ensuring that authorization, auditing, and escalation protocols are rigorously enforced across all agent-managed development.

Key risks: The paramount risk at this stage is total opacity: no single human can comprehend the full state of the system. Without strict governance, authorization boundary violations will cascade rapidly at scale, and any gaps in audit trails will make incident response effectively impossible.

The next step is managing multiple AI agents operating autonomously at scale. SCW training covers the agent authorization framework, predictive risk scoring and provenance linking. Trust Agent provides authorization enforcement, audit trails and auto training assignments.

PHASE: AGENTIC

8 Autonomous orchestration

Risk level: maximum

Organizational indicators: At this ultimate stage of autonomous orchestration, the paradigm shifts entirely: master agents autonomously direct other agents to execute the complete software engineering lifecycle, from research and design to coding, testing, documentation, and deployment. Architecture is no longer driven by human decisions on a per-feature basis; instead, it is expressed purely as an overarching policy. The system becomes entirely self-directing within these defined boundaries, with human involvement strictly reserved for high-risk escalation points.

Developer experience: Your daily work evolves into pure governance. Instead of reviewing code or orchestrating tasks, you spend your time defining "secure" in machine-readable terms and establishing the policies and constraints that the autonomous system operates within. The system operates continuously in the background, leveraging your expertise in risk frameworks and policy definition. You intervene only when explicitly notified of high-risk exceptions that require human approval.

The shift to spec-driven development: The software engineering landscape is fundamentally changing with the rise of "spec-driven development", widely considered the future of how software is built. Instead of writing line-by-line syntax, developers will work alongside AI agents to translate intent into execution. This shifts the engineering role from a traditional coder to a holistic creator, blending the craft of product management, design thinking, architectural oversight, and code orchestration into a single, unified position. Success in this new paradigm will belong to those who can masterfully articulate, structure, and guide AI agents through precise, high-level specifications.

Security posture and practices: Security at this stage is entirely policy-driven, necessitating a flawless trust framework. "Policy-as-code" dictates that all security requirements be machine-readable and strictly enforceable. As human-in-the-loop interventions occur only at explicitly defined escalation points, organizations must mandate comprehensive, automated SBOM generation and continuous compliance monitoring via automated attestation. Teams must, in addition, develop novel incident response protocols designed specifically for autonomous systems, underpinned by rigorous AI code provenance that maintains a full audit trail of exactly which agent and model produced every line of code.

Key learning concepts: The foundational primitive of this stage is "policy-as-code", or encoding security requirements as absolute runtime constraints rather than documented wiki guidelines. Teams must establish risk scoring as a platform primitive, automatically gating deployments by attaching live risk scores to every change based on provenance, vulnerability signals, and historical patterns. This stage demands mastering AI code provenance at scale, effectively creating a tamper-evident SBOM (or AIBOM) for the generative layer. Developers must also design technically enforced "human-in-the-loop" escalation points that resist alert fatigue, learn forensic incident response for AI attack surfaces, and directly confront the deep legal, ethical, and technical tensions of accountability: who is ultimately responsible for the security of autonomous output?

Primary security responsibility: Your ultimate mandate is to define, maintain, and continuously evolve security policies in machine-readable form, establish ironclad escalation criteria, and ensure that governance frameworks keep pace with rapidly advancing autonomous capabilities.

Key risks: This stage carries maximum risk. If policy rules are flawed, the autonomous system will systematically introduce vulnerabilities at an unprecedented scale. Additional catastrophic risks include policy drift as AI capabilities evolve autonomously, and the complete loss of reliable auditability

In terms of how this final stage looks in the real world, OpenAI's AI-native Software Development Life Cycle (SDLC) framework redefines traditional engineering into an accelerated, four-stage loop consisting of **Plan, Build, Review, and Deploy**. In this method, AI agents evolve from simple autocomplete assistants into contextual workflow partners that analyze codebases to map technical specifications during the Plan phase, drive parallel code generation in the Build phase, execute automated policy and error checks during Review, and optimize continuous integration pipelines for final Deployment. By implementing a "Delegate, Review, Own" philosophy, this end-state model maximizes engineering velocity while ensuring that (security-skilled) human developers retain critical oversight over architectural integrity, security validation, and final code ownership.

SCW training covers AI code provenance as a platform, risk scores in the deployment pipeline and policy-as-code. Trust Agent provides full-stack observability, CISO dashboards, and live LLM benchmarking insights.

Most developers are currently working somewhere in **Stages 2 through 4**. At these stages of their AI journey, the productivity gains from using AI are very clear, but the governance gaps at those stages are becoming critical, as evidenced by the High risk levels of **Stages 3 and 4**.

Overall, **35% to 45%** of developers are at **Stages 1 and 2**, performing basic tasks with AI while gradually increasing oversight. A roughly equal number **(35-45%) are at Stages 3 and 4**, where the going gets more treacherous. There is less oversight at these stages, where AI-native tools operate more autonomously, and risk levels increase precipitously. Only **10% to 18%** of developers are operating at either Stage 5 (an AI native category with a Very High risk level) or the Agentic AI stages of 6 through 8, where risk levels push the envelope and the need for governance is critical. But that cohort figures to grow quickly as more developers pursue AI's capabilities. The largest contingent of developers is expected to be working at **Stages 3 through 5** by the end of 2026.

35% to 45% Developers at Stages 1 & 2 performing basic tasks

35% to 45% Roughly equal number at Stages 3 & 4

10% to 18% Developers operating at either Stage 5, or 6 through 8

SCW training covers AI code provenance as a platform, risk scores in the deployment pipeline and policy-as-code. Trust Agent provides full-stack observability, CISO dashboards, and live LLM benchmarking insights.

Most developers are currently working somewhere in **Stages 2 through 4**. At these stages of their AI journey, the productivity gains from using AI are very clear, but the governance gaps at those stages are becoming critical, as evidenced by the High risk levels of **Stages 3 and 4**.

Overall, **35% to 45%** of developers are at **Stages 1 and 2**, performing basic tasks with AI while gradually increasing oversight. A roughly equal number **(35-45%) are at Stages 3 and 4**, where the going gets more treacherous. There is less oversight at these stages, where AI-native tools operate more autonomously, and risk levels increase precipitously. Only **10% to 18%** of developers are operating at either Stage 5 (an AI native category with a Very High risk level) or the Agentic AI stages of 6 through 8, where risk levels push the envelope and the need for governance is critical. But that cohort figures to grow quickly as more developers pursue AI's capabilities. The largest contingent of developers is expected to be working at **Stages 3 through 5** by the end of 2026.

Further reading on risk inflections: the two stages where risk levels spike

Obviously, risk levels increase in step with the autonomy and capabilities of AI systems — the more an AI agent does, the higher the risk that something could go sideways — but there are two critical inflection points along the way that change the game. You need to pay particular attention to them.

The first risk inflection point happens at Stage 3. This is where AI agents leave the nest, so to speak, moving from supervised to unsupervised activity. Developers grant the agent broad permissions and stop carefully reviewing its output. As trust in the tool increases, oversight decreases, which can lead to missteps such as deploying vulnerable code to production. Training that covers observability and the identification of over-trust patterns, along with a tool that detects incidents in which an agent is overtrusted, can help ensure the effectiveness of a governance policy.

The second inflection point occurs at Stage 6, when multiple agents work in parallel. At this point, the developer is simultaneously orchestrating, say, three to five agents that are generating code at scale. Individual code review at this stage is physically impossible. Governance must shift from human effort to system-level control, with risk management being entirely policy-driven. Developers need to be conversant with the policy enforcement engine and able to track application risk scorecards and trace commits so they can be compiled for audits. A tool like Trust Agent can also provide container support and cross-agent traceability.

Aside from the inflection points, security teams need to be aware of the progression from AI-Assisted, where humans are still in charge of writing code, to AI-Native, where AI takes over most of the code-writing, tests and delivery artifacts, to Agentic, where agents work in concert out of the control of overseers.

As the stages progress, five things occur:

- Velocity increases
- Human authorship decreases
- Risk becomes harder to detect
- Cost scales with usage
- Accountability is becoming more complex

Together, those factors present a challenge that can't be ignored (or even put off for a while).

The cost crisis: why CISOs need to act now

By now, the urgency posed by agentic AI's growing presence in the enterprise should be clear, particularly regarding security. Gartner's 2026 Hype Cycle for Secure Software Engineering puts it plainly: AI coding tools are making secure coding skills more important than ever to ensure the security of AI-generated code[2]. The organizations that treat developer upskilling as a cost center rather than a governance control will pay for it downstream, in breach costs, remediation cycles, and technical debt that compounds at the speed of AI output. But CISOs have other reasons to adopt a governance policy without delay. One of those progressively significant reasons is cost.

As organizations move up the adoption curve and increase their use of AI, costs will scale with it, sometimes in unexpected (and expensive) ways. Prices for AI models can vary widely, depending on the required capabilities, ranging from free open-source models to global enterprise models that can cost more than **$1 million** as a one-time fee or charge a hefty monthly subscription fee.

The mushrooming use of AI also gets more expensive as it grows, with costs incurred via regeneration and iteration cycles, parallel agent activity and unnecessary work created by AI. Another hidden cost results from token use, which is charged when an agent requests processing from a GenAI model. Input charges can range from **15 cents to $5 per million**, while output charges are between **60 cents and $25 per million**. With constant autonomous activity from AI agents, those costs can mount quickly, especially if that activity is unsupervised.

Input charges can range from 15 cents to $5 per million, while output charges are between 60 cents and $25 per million. With constant autonomous activity from AI agents, those costs can mount quickly, especially if that activity is unsupervised.

Another factor to consider is the quality of the models available. Our benchmarking of the leading models shows a wide spread in secure-code performance, from highs in the **mid-60%** of secure code generated to **lows around 30%**. CISOs need to carefully choose the models they employ and ensure they are suited to the tasks at hand. The research also found no reliable correlation between model cost and security outcomes. The most expensive model per run costs nearly 8X as much as the highest-scoring model for a comparable normalized security score. Meanwhile, the cheapest models deliver dramatically worse security outcomes, suggesting they may suit cost-sensitive applications only where security can be addressed downstream.

Given the variance in model prices and the costs of running AI systems, CISOs must face the new reality that cost, quality, and risk are now inseparable. Without effective governance, they open the door to an unsustainable environment of unbounded AI use, unpredictable costs, and growing inefficiency.

Generative AI and agentic AI are radically changing the SDLC, whether CISOs like it or not, to the point that the SDLC, which has been the foundation of software development for decades, is no longer valid. The age of the Agentic Development Lifecycle is upon us.

[2]Gartner Hype Cycle for Secure Software Engineering (2026). Harrison, A.

The SDLC has been undeniably beneficial, but it is a static, structured process for building, testing, deploying and maintaining software according to a schedule. In recent years, DevOps and agile development have chipped away at the idea of a schedule by enabling faster application development and deployment, but agentic AI is demolishing established processes altogether. The ADLC is designed for a dynamic environment in which autonomous agents act independently and evolve over time. The ADLC is still structured, but is also adaptive. Enterprises need to ensure that they can also be governed.

The approach to managing risk also needs to shift dramatically. With the rise of agentic AI, proactive governance throughout the process is imperative as the SDLC transitions to the ADLC. SCW has recognized this critical need by prioritizing AI software governance.

Conclusion

As the use of agentic AI for software development grows exponentially, many enterprise security leaders are struggling not just with the inherent risks of AI-generated code, but also with a compounding set of challenges: shadow AI, the poor observability of AI tools in use, and low confidence in the security proficiency of developers and employees using AI, all while vast swathes of AI-generated code go into production. In this new environment, traditional approaches to risk management and governance are breaking down. CISOs need to take a new approach to governance, recognizing that the ADLC must replace the SDLC, and following an adoption model designed for agentic AI's evolving, adaptive approach to software development.

By mapping each stage of AI adoption to the right combination of developer upskilling and governance controls, this AI Adoption Model provides CISOs and security leaders with a practical framework for governing AI usage within an organization, especially as software development shifts from assistance to autonomy. By linking each stage of AI adoption to the right combination of observability, policy controls and contributor upskilling, the model helps organizations identify emerging risk, strengthen oversight and build governance into the ADLC before gaps widen.

The SCW Learning Platform contains tailored content mapped precisely to each stage of the developer's growth journey. Rather than relying on manual assessments, the platform automatically classifies developers into their respective stages by analyzing telemetry and signals directly from their machines, development tools, and code repositories. This behavioral insight allows organizations to seamlessly push the right educational content to the right developer at the exact moment they need it, maximizing engagement and relevance.

In an environment where AI adoption is outpacing most enterprises' ability to track it, such a structured framework can help turn AI governance from a reactive exercise into a measurable, scalable discipline.

As the use of agentic AI for software development grows exponentially, many enterprise security leaders are struggling not just with the inherent risks of AI-generated code, but also with a compounding set of challenges: shadow AI, the poor observability of AI tools in use, and low confidence in the security proficiency of developers and employees using AI, all while vast swathes of AI-generated code go into production.