PARADIGM · WHITEPAPER · No. 01

The Enterprise Trust Platform for AI Driven Software Development

Executive Summary

Artificial intelligence is fundamentally transforming how software is created. Software engineering is shifting from a human-centric process to one increasingly driven by AI coding agents that enable both professional developers and non-developers to create software, applications, and business automations. Coding agents understand requirements, generate and modify code, review software, execute tests, and participate throughout the software development lifecycle. However, research has shown that even advanced coding models can introduce security vulnerabilities into the software they generate[1]. As enterprises adopt multiple coding agents across their development organizations, a fundamental new question emerges: how do organizations establish trust in software that is increasingly created by AI? Four challenges become particularly important.

First, organizations must govern AI coding agents themselves. Coding agents can access enterprise data and repositories, use foundation models, invoke development tools and MCP servers, execute actions, and consume significant AI resources. Without centralized governance, every coding agent effectively becomes an independent software engineering participant with its own access, capabilities, behavior, and token consumption. Enterprises need consistent security, access, execution, observability, and cost controls across coding agents, including policies governing model usage, tokens, quotas, and budgets.

Second, organizations must ensure that every coding agent creates software using the same enterprise knowledge and requirements. Development policies, secure coding standards, architectural requirements, mandated skills, and regulatory requirements must be applied consistently regardless of which coding agent or model is used. Equally important, organizations accumulate valuable security knowledge from previous vulnerabilities, penetration tests, security incidents, code reviews, architectural issues, and remediation. This institutional knowledge should inform future software creation so that coding agents learn from and do not repeat the mistakes of the past.

Third, organizations must independently verify the software generated by AI coding agents. Providing the right instructions does not guarantee the right result. AI-generated software must be verified to ensure that it implements developer intent, applies required enterprise policies and skills, satisfies secure coding and organizational security requirements, and does not repeat previously identified security issues before it becomes part of the software supply chain.

[1] H. Liu, L. Einstein, J. Yang, J. Baumann, D. Eddy, C. D. Manning, M. Kochenderfer, and D. Yang, *SecureForge: Finding and Preventing Vulnerabilities in LLM-Generated Code via Prompt Optimization.*

Finally, organizations must preserve the complete memory of how software was created and evolved. Source code alone does not capture developer intent, prompts, models, tools, policies, security knowledge, or verification evidence that influenced its creation. And software creation does not end when code reaches the repository; it continues through pull requests, reviews, builds, dependencies, SBOMs, software artifacts, CI/CD pipelines, deployment, security findings, remediation, and subsequent AI-driven modifications. Enterprises therefore need a persistent record connecting why software was created, how it was created and verified, and how it evolved throughout its lifecycle.

AI - Driven Software Development Requires a New Trust Architecture

For decades, software was designed, written, tested, and deployed primarily by people, and enterprise development and security processes were built around this human-centric model. AI coding agents fundamentally change this. As agents increasingly participate throughout the software lifecycle, enterprises need a common trust architecture that governs the agents creating software, guides how software is created, verifies what is generated, and maintains its provenance and integrity throughout its lifecycle.

Paradigm: The Trust Architecture for AI-Driven Software Development

Paradigm (Figure 1) provides a unified platform for establishing trust across the AI Software Development Lifecycle (AI SDLC). At its foundation is CodeChain™, the Enterprise Trust Graph for AI-Generated Software, which connects governance, enterprise knowledge, verification, security learning, and software provenance into a persistent record of how software is created and evolved. Paradigm delivers four capabilities for establishing trust across the AI SDLC. Three of them—governing coding agents, guiding software creation, and verifying what is generated—continuously write their evidence into the fourth, CodeChain, the trust graph that connects them and preserves the memory of how software was created:

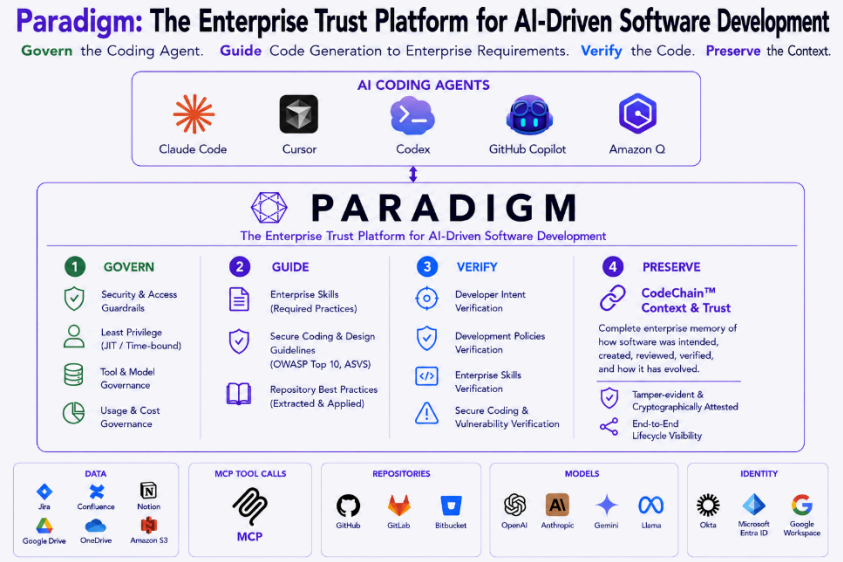


Figure 1: Paradigm Enterprise Trust Platform for AI-Driven Software Development

Govern AI Coding Agents

Paradigm provides a common governance layer across coding agents, ensuring they operate within enterprise-defined boundaries regardless of which coding agent or foundation model developers choose. Paradigm governs who can use an agent, what the agent can access, which models, tools and MCP servers it can use, what actions it can perform, and how much it can consume. This includes identity and access, enterprise data and repositories, runtime permissions, prompt security, execution controls, observability, and auditability. Paradigm also provides visibility and policy controls over model usage, token consumption, quotas, and budgets, allowing enterprises to manage the cost of increasingly autonomous software development.

Guide Software Creation with Enterprise Knowledge

Paradigm ensures that coding agents consistently apply the organization's development policies, architectural standards, secure coding requirements, mandated skills, regulatory requirements, and engineering practices during software generation, regardless of which coding agent or model is being used. Paradigm also incorporates institutional security knowledge from previous vulnerabilities, penetration tests, security incidents, code reviews, architectural issues, root-cause analysis, and remediation. What the enterprise learns from software today can therefore become guidance, policy, or a required skill for software created tomorrow, helping prevent previously discovered problems from recurring.

Verify AI-Generated Software

Paradigm independently verifies software generated by coding agents before it progresses through the software lifecycle. Verification determines whether the software generated implements developer intent, applies required enterprise policies and skills, follows architectural and secure

coding standards, satisfies organizational security requirements, and does not repeat known security problems. When verification fails, Paradigm can provide corrective guidance, require regeneration, or prevent the software from progressing. This provides an independent check between what the coding agent was instructed to do and what it produced.

CodeChain™: The Enterprise Trust Graph for AI-Generated Software

CodeChain is the foundational execution and trust graph within Paradigm, creating the persistent context and enterprise memory of how software was intended, created, reviewed, verified, and evolved throughout its lifecycle. A CodeChain is associated with a software artifact or logical software entity, such as an application, service, agent, or component, and can connect multiple repositories, components, builds, and deployments that collectively make up that software. It captures three aspects of context:

Governance Memory: What Was Supposed to Happen. Captures developer intent, enterprise policies, secure coding requirements, mandated skills, approved models and tools, permissions, cost controls, and governance decisions that shaped software creation.

Security Memory: What the Enterprise Has Learned. Captures institutional knowledge from vulnerabilities, penetration tests, security incidents, code reviews, root causes, and remediation. These learnings can become new guidance, policies, skills, and verification requirements, allowing what the enterprise learns today to improve how software is created tomorrow.

Provenance & Software Integrity Memory: What Actually Happened. Captures how software was created and evolved, from developers, prompts, agents, models, tools, and generated code through verification, commits, pull requests, code reviews, builds, SBOMs, artifacts, deployments, and subsequent changes. The results of code reviews and pull requests, including changes made in response to them, become part of the CodeChain, preserving the evolution of the software and the decisions that shaped it.

CodeChain™ is checked into and maintained alongside the software in the source code repository, allowing its trust context to travel with the software as it evolves. Cryptographic attestations bind CodeChain evidence to the corresponding software artifacts, providing tamper-evident provenance and enabling enterprises to verify that the software and its recorded history have not been altered without detection. Together, these memories form CodeChain's continuously evolving Enterprise Trust Graph, preserving the context of what the enterprise intended, what governed software creation, what happened, what was reviewed and verified, and what the enterprise learned. This persistent enterprise memory follows the software across repositories, components, builds, and deployments, providing the context needed to understand, review, verify, secure, and securely evolve software throughout its lifecycle.

Enterprise Outcomes

Paradigm enables enterprises to confidently adopt and scale AI-driven software development by:

- **Accelerating enterprise adoption of AI coding agents:** Give developers the freedom to use the coding agents and models that best meet their needs, while giving security and engineering leaders confidence that AI-driven development remains controlled, secure, and accountable.
- **Improving software quality and reducing security risk:** Create software that more consistently follows enterprise standards, incorporates institutional security knowledge, and is independently verified—reducing vulnerabilities, policy violations, and the recurrence of previously identified problems.
- **Reducing the cost and operational risk of AI-driven development:** Gain visibility into how coding agents, models, and tokens are being used across the organization and control AI spend through enterprise-wide usage, quota, and budget policies.
- **Creating lasting trust and accountability for AI-generated software:** Maintain a persistent and verifiable history of why software was created, how it was created and verified, and how it evolved through the software supply chain—providing the evidence and context needed for security, compliance, audit, incident response, and future software development.

As AI coding agents become increasingly central to software development, enterprises should not have to choose between the speed and productivity of AI and the security, control, and accountability required for enterprise software. They need the freedom to adopt the best coding agents and models while maintaining a consistent layer of trust across the entire AI software development lifecycle. Paradigm provides that independent trust layer, enabling enterprises to embrace AI-driven software development with any coding agent with confidence and create software they can trust, regardless of which coding agent, model, or development environment they choose.