ETIS WHITE PAPER SERIES

WP-001

Engineering Trustworthy Software in the AI Era

From Code Generation to Governed, Agentic, and Evidence-Centered Delivery

Core Thesis

Artificial intelligence does not eliminate software engineering. It raises the standard. As intelligent systems gain the ability to generate, reason, coordinate, and act, the engineer's central responsibility shifts from producing artifacts to controlling intent, context, authority, change, evidence, and operational behavior.

William T. O'Connell, Ph.D.

Adjunct Professor of Computer Science, Loyola University Chicago
Former IBM Distinguished Engineer and CTO, IBM Consulting Quality and Delivery

ETIS White Paper WP-001 | Revised Publication Edition | July 2026

Abstract

Generative and agentic artificial intelligence are changing the economics, tempo, and control surface of software delivery. The first wave made code easier to produce. The emerging wave delegates bounded engineering work to agents that can inspect repositories, formulate plans, modify code, run tools, execute tests, prepare pull requests, and coordinate long-running tasks. This progression is not the end of software engineering. It is a redistribution of effort away from manual artifact production and toward specification, architecture, context, verification, governance, and operational control.

This paper argues that trustworthy intelligent systems cannot be engineered by treating a model—or the code it generates—as the system. Trust depends on the complete sociotechnical environment: purpose, requirements, architecture, data, context, identity, permissions, tools, software supply chain, tests, evaluations, release controls, observability, human oversight, and the evidence connecting them. The industry is converging on this conclusion from several directions. DORA describes AI as an amplifier of organizational capability and dysfunction. NIST and ISO are moving responsible AI from principles toward management systems, impact assessment, secure development, and assurance. OWASP is extending application security to agent goals, tools, identity, memory, and cascading action. Development platforms are evolving from copilots into governed engineering agents.

The paper presents a practical synthesis: repository-centered and evidence-centered engineering organized around three layers of governance and four levels of AI autonomy. It examines near-term product and methodology directions, explains how engineering and leadership roles will change, and positions Engineering Trustworthy Intelligent Systems (ETIS) as an integration model connecting established software engineering, modern delivery practices, AI governance, agentic security, and operational stewardship. It is written for executives, engineering leaders, architects, practitioners, educators, and students.

Keywords—software engineering, agentic AI, trustworthy intelligent systems, repository-centered engineering, evidence-centered engineering, context engineering, AI governance, DevSecOps, operational readiness, human oversight

Executive Summary

Software engineering is entering a transition as consequential as the adoption of the Internet, cloud computing, mobile platforms, and DevOps. The visible symptom is AI-generated code. The deeper change is the movement of AI into discovery, planning, design, implementation, testing, modernization, deployment, operations, and enterprise workflow execution. OpenAI, GitHub, IBM, and AWS now describe products or methods that operate across substantial portions of the development lifecycle rather than within a single editor interaction.[2, 3, 4, 5]

The strategic implication is often misunderstood. When software artifacts become cheaper and faster to produce, engineering discipline does not become less important. It becomes the limiting factor. Ambiguous requirements can be implemented faster. Weak architecture can decay faster. Insecure dependencies can spread faster. Poorly reviewed changes can reach production faster. DORA's research characterizes AI as an amplifier: it magnifies the strengths of capable organizations and the dysfunctions of struggling ones, improving throughput while threatening stability where the surrounding system is weak.[1]

For executives, the issue is therefore not simply developer productivity. It is whether the enterprise can convert increased production capacity into trustworthy outcomes. That requires an operating model in which intent is explicit, authority is bounded, work is reviewable, evidence is preserved, controls are enforced at design time and runtime, and operational learning feeds back into the next change. For engineers, the craft moves upward: from typing code toward defining specifications, shaping context, designing boundaries, supervising agents, challenging results, and defending release decisions.

The executive question

The central question is no longer, “How much code can AI produce?” It is, “How much trustworthy change can the organization absorb, verify, govern, and operate?”

This paper advances five conclusions. First, the model is not the system; intelligent behavior emerges from the interaction of models with data, tools, permissions, workflows, software, and people. Second, context is a controlled engineering asset, not an informal prompt. Third, governance must exist at design time, delivery time, and runtime. Fourth, every meaningful engineering claim should be supported by inspectable evidence. Fifth, autonomy must increase only when authority controls, observability, intervention, and accountability increase with it. These conclusions form the intellectual foundation of ETIS.

1. The Inflection Point: From Coding Assistance to Delegated Delivery

The first generation of AI coding tools improved local productivity. They completed statements, generated functions, explained unfamiliar code, and reduced the friction of routine implementation. The next generation is materially different. GitHub’s cloud coding agent can research a repository, create a plan, modify code on a branch, and open a pull request for human review. OpenAI describes Codex as capable of completing pull requests, refactors, migrations, code reviews, and parallel background work. IBM positions Bob as an AI-first development partner spanning planning, coding, testing, deployment, modernization, governance, and security. AWS’s AI-Driven Development Life Cycle places AI throughout inception, construction, and operations rather than confining it to coding. [2, 3, 4, 5]

This is a shift from assistance to delegation. Assistance occurs when a human remains inside the immediate task loop and uses AI to accelerate a step. Delegation occurs when a human defines an outcome, supplies constraints and context, and allows an agent to execute a bounded sequence of actions before returning work for review. Delegated delivery can create enormous leverage, but it also changes the engineering risk. The agent may choose files, tools, dependencies, commands, and implementation paths that were not individually specified. Review must therefore evaluate not only the output but also the assumptions, authority, and path by which the output was produced.

The product direction is becoming clear. Engineering environments are moving toward multi-agent work, specialized agent profiles, persistent project instructions, repository-aware context, asynchronous execution, and integrated review. GitHub supports specialized custom agents and agent skills. OpenAI describes coordinated teams of coding agents working across design, building, shipping, and maintenance. IBM has announced multi-agent capabilities and specialized modernization workflows. These developments suggest that the near-term engineering environment will look less like a developer paired with a single assistant and more like a human-led delivery system in which several agents perform bounded roles. [6, 7, 8]

That environment changes the bottleneck. Code generation becomes abundant. High-quality intent, architecture, constraints, test oracles, production knowledge, and review capacity remain scarce. Organizations that fail to strengthen those capabilities may experience synthetic productivity: more output, more activity, and more change, without a proportional increase in reliable business value.

A four-level autonomy model

It is useful to distinguish four levels of AI participation because each level creates a different control burden. The progression is not a maturity score; it is a transfer of authority.

Level	Engineering role of AI	Control implication
Assist	Suggests, explains, drafts, or generates within a human-controlled task.	Human reviews every meaningful output before use.
Coordinate	Organizes work, proposes plans, manages context, and sequences lifecycle activities.	Plans, dependencies, and acceptance conditions must be explicit and reviewable.
Execute	Performs bounded multi-step engineering work using repositories, tools, and test environments.	Authority, sandboxing, provenance, testing, and approval gates become first-class controls.
Operate	Takes action in or on behalf of a live system.	Runtime policy, least privilege, observability, intervention, rollback, and accountable ownership are mandatory.

The industry is already moving through the first three levels, and selected enterprise platforms are moving toward the fourth. The appropriate response is neither enthusiasm without controls nor prohibition without learning. It is deliberate engineering: expand authority only when the system can demonstrate that it has the context, controls, evidence, and human oversight necessary for the consequences of that authority.[9, 10]

2. What Changes—and What Does Not—in Software Engineering

The software development lifecycle is not disappearing. Its durable responsibilities remain: discover the real problem, define requirements, design a coherent system, implement change, verify behavior, control configuration, release responsibly, operate reliably, and learn from failure. AI changes how those responsibilities are performed and raises the importance of several that were previously underfunded.

Requirements become more than descriptions of features. In intelligent and agentic systems, they must express desired outcomes, prohibited actions, data and tool boundaries, human approval points, acceptable uncertainty, failure behavior, and evidence of success. A requirement increasingly functions as an operational contract among product owners, engineers, agents, security teams, operators, and affected users. If intent remains vague, AI can implement the wrong interpretation with unprecedented speed.

Architecture returns to the center of delivery. When AI can create code quickly, local implementation decisions become easier to make and harder to govern. Boundaries, ownership, interfaces, trust zones, data flow, agent authority, context sources, observability, and failure containment must be established early enough to constrain accelerated production. OpenAI's own account of building with coding agents emphasizes that architectural constraints become an early prerequisite because they allow speed without decay or drift.[11]

Testing expands from deterministic correctness to behavioral assurance. Conventional unit, integration, regression, performance, and security testing remain necessary. Intelligent behavior additionally requires evaluation of choices, tool use, policy compliance, information leakage, escalation, abstention, recovery, and side effects. The test oracle may be a distribution, rubric, policy, or human judgment rather than a single expected value. Verification must therefore combine automated tests, evaluation datasets, adversarial scenarios, runtime telemetry, and review by people who understand the domain.

Deployment also changes. The release unit may include source code, prompts, system instructions, retrieval configuration, model and routing settings, tool permissions, policies, evaluation assets, monitoring rules, and operating procedures. Each independently changeable element needs ownership, versioning, validation, and rollback thinking. A system can change behavior even when its application code has not changed.

A durable engineering rule

Every independently changeable element that can alter system behavior should have an owner, a controlled version, a validation method, and a recovery path.

Operations become the place where trust is continuously tested. Traditional telemetry—availability, latency, errors, saturation, and resource use—must be joined by quality signals, tool calls, policy violations, human overrides, context failures, cost, drift, and business outcomes. The purpose is not to collect more dashboards. It is to preserve the ability to understand what the system did, why it did it, whether the behavior remained within authority, and what must change next.

3. The System Is Larger Than the Model

Many AI initiatives begin with model selection and treat the surrounding application as implementation detail. This reverses the engineering problem. The model is a probabilistic component embedded in a larger sociotechnical system. That system includes authoritative data, retrieval and context mechanisms, software services, identities, permissions, tools, workflows, user experience, policies, monitoring, operators, and decision owners. Trust is a property of how those elements interact—not a property that can be assigned to the model in isolation.[12]

This distinction becomes critical when models can act. A conversational error may mislead a user. An agentic error can alter a record, invoke an API, approve a transaction, delete data, publish code, or trigger a chain of downstream actions. The risk is no longer only a wrong answer. It is a wrong action executed with real authority.

Context engineering as a control discipline

Context engineering is often described as giving an AI system the information it needs. In professional engineering, that definition is incomplete. Context also limits what the system can infer and do. Requirements, architecture records, coding standards, repository maps, prior decisions, domain rules, runbooks, policies, known defects, test expectations, and production telemetry all shape agent behavior. Context therefore has the same characteristics as other controlled engineering assets: provenance, scope, freshness, access, ownership, review, and change history.

This is one reason the repository is becoming more important, not less. A well-maintained repository carries the explicit memory of the engineering organization. It contains not only source code but also intent, decisions, controls, tests, release records, operational guidance, and evidence. For humans, this improves review and continuity. For agents, it supplies structured context. For governance, it creates an inspectable record of why the system exists, how it changed, and what supported the decision to release it.

The implication for leaders is direct: documentation quality, architecture discipline, and repository hygiene are now productivity infrastructure. They are no longer secondary administrative concerns. An

organization that cannot explain its systems to its own engineers will struggle to provide reliable context to autonomous engineering agents.

4. Governance Becomes Architecture and Operating Discipline

Traditional governance often appears as a sequence of external reviews: legal, security, compliance, architecture, and release approval. That model is too slow and too discontinuous for systems whose behavior can change through models, context, policies, data, or tools. Governance must move into the design and operation of the system.

NIST's AI Risk Management Framework organizes risk work around Govern, Map, Measure, and Manage, and its Generative AI Profile extends that guidance to generative systems. ISO/IEC 42001 establishes an organizational management system for responsible AI, including policy, objectives, risk management, lifecycle controls, performance evaluation, and continual improvement. ISO/IEC 23894 addresses AI risk management, while ISO/IEC 42005 adds a formal direction for AI system impact assessment. These standards do not prescribe a particular repository or delivery workflow. Their significance is that they move AI responsibility from voluntary principles toward repeatable management, evidence, assessment, and assurance.[12, 13, 14, 15]

Software engineering must operationalize those expectations. A useful model has three connected layers. Design-time governance defines purpose, authority, prohibited behavior, data boundaries, architecture, risk, and human oversight. Delivery-time governance controls changes through issues, branches, reviews, CI/CD, testing, provenance, approvals, and release evidence. Runtime governance enforces policy, identity, permissions, monitoring, intervention, rollback, and audit. Weakness in any one layer undermines the others.

Governance layer	Primary question	Typical engineering evidence
Design time	What may the system do, under what constraints, and who owns the consequences?	Requirements, architecture, threat model, impact assessment, authority matrix, ADRs.
Delivery time	Why is this change safe enough to integrate and release?	Issue, branch, pull request, review, tests, CI/CD results, provenance, release record.
Runtime	Is the deployed system behaving within authority, and can we intervene?	Telemetry, audit logs, policy decisions, alerts, human overrides, rollback and incident records.

Agentic security illustrates why this integration is urgent. OWASP's 2026 Top 10 for Agentic Applications identifies risks such as goal hijacking, tool misuse, identity and privilege abuse, memory poisoning, insecure inter-agent communication, cascading failures, and rogue agents. These are not merely model-quality problems. They are architecture, identity, application security, workflow, and operational-control problems.[16, 17]

Governance is not a document

A policy that cannot be enforced, observed, tested, and audited is an aspiration. In intelligent systems,

governance becomes real only when it is implemented through architecture and operating discipline.

5. Repository-Centered and Evidence-Centered Engineering

The repository began as a place to manage source code. In modern delivery it has become a coordination surface for intent, implementation, review, automation, and release. In AI-assisted delivery it becomes something more consequential: the engineering system of record from which humans and agents derive context and in which the organization preserves evidence.

Repository-centered engineering does not mean that every corporate record belongs in Git. It means that the engineering claims necessary to understand, review, reproduce, and operate a system should be connected to versioned and inspectable evidence. Requirements explain why work exists. Issues turn intent into accountable activity. Architecture records preserve decisions and boundaries. Pull requests expose proposed change and review. Automated checks provide verification. Release records define a baseline. Runbooks, postmortems, and operational evidence preserve what was learned after deployment.

The pull request is a useful example. In a weak workflow it is a merge button. In a professional workflow it is a compact engineering argument: what changed, why the change was needed, what assumptions were made, how the change was tested, what risks remain, how AI was used, what reviewers challenged, and why the team considered the change safe enough to integrate. As agents produce more changes, that argument becomes more valuable than the raw diff.

Evidence-centered engineering extends the idea. Engineering decisions are claims: the requirement is understood; the architecture is suitable; the code is correct enough; the release is ready; the system can be operated; the risk is accepted. Trustworthy teams connect those claims to evidence and preserve the chain from intent through operation. The objective is not artifact volume. It is the ability of another competent person to examine the record and reach a defensible conclusion.

Supply-chain integrity reinforces this direction. NIST's Secure Software Development Framework defines fundamental practices that should be integrated into any SDLC. SLSA focuses on provenance and protection against tampering in the software supply chain. Together with version control, CI/CD, and release evidence, these practices make it increasingly possible to show not only what was delivered but also how it was produced and protected.[18, 19]

6. The Enterprise and Consulting Agenda

Enterprise clients are not primarily asking for better chatbots. They are asking how AI can redesign value streams, modernize aging estates, improve service, increase engineering capacity, reduce risk, and create measurable operating advantage. The demand is moving from isolated experimentation toward governed platforms, shared context, reusable controls, agent orchestration, and operating models that can scale across business units.

Technology products reflect that demand. IBM Bob emphasizes enterprise software delivery and modernization across the SDLC. ServiceNow's AI Control Tower focuses on discovering, governing, securing, and measuring agents, models, and workflows. Salesforce, SAP, UiPath, Microsoft, and other platform vendors are investing in agent orchestration tied to systems of record and business processes. The common

direction is not one universal agent. It is an enterprise control environment in which specialized agents act through governed tools and data.[4, 20, 21, 22]

Consulting firms are correspondingly shifting their focus. The work is less about installing a model and more about redesigning the operating model: selecting value streams, clarifying decision rights, building data and platform foundations, modernizing applications, embedding security and governance, creating reusable delivery patterns, and measuring realized value. The difficult work crosses organizational boundaries. Business leaders, product owners, architects, engineers, security, legal, risk, operations, and workforce leaders must agree on what the system is allowed to do and how success will be judged.

Different industries emphasize different consequences. Financial institutions focus on model risk, transaction integrity, auditability, fraud, and regulatory accountability. Healthcare organizations emphasize safety, privacy, clinical responsibility, and the integrity of human decisions. Government agencies require transparency, public trust, procurement discipline, records, and continuity. Manufacturers and transportation providers focus on safety, cyber-physical consequences, resilience, and supply chains. Universities and professional-services organizations confront data governance, intellectual property, quality, and workforce transformation. The common engineering need is the same: integrate AI without losing control of the system or accountability for the outcome.

This is also why executives should resist productivity narratives based only on generated code, completed tickets, or reduced labor. DORA's findings point toward a more balanced view in which throughput and stability must be considered together. The business outcome is not more software; it is a better-performing, more adaptable, and more trustworthy operating capability.[1]

7. The Changing Work of Engineers and Leaders

AI will compress some routine work and increase the leverage of people who can exercise judgment. The vulnerable tasks are those that transform clear instructions into standard output: boilerplate implementation, simple test scaffolding, repetitive documentation, low-context analysis, and mechanical ticket translation. Those tasks will not disappear uniformly, but their economic value will decline as agents become more capable.

The valuable work moves toward ambiguity, architecture, integration, verification, security, operations, and accountability. Developers will spend more time formulating work, reviewing generated changes, designing tests, diagnosing failures, and maintaining context. Senior engineers will define boundaries, prevent architectural drift, and govern agent-produced change. Quality engineers will design evaluation systems for nondeterministic behavior. Security professionals will control tool authority, identities, data flow, and agent interaction. Product leaders will define outcomes, autonomy boundaries, and intervention points. Operators will monitor behavior, cost, quality, and policy compliance—not only uptime.

Leadership must also change. Traditional capacity planning assumes that implementation effort is scarce. AI may make implementation capacity abundant while review, architecture, domain expertise, and operational absorption remain constrained. Leaders who simply increase the volume of work in progress will create queues and instability. The stronger response is to reduce batch size, improve specifications, strengthen platform capabilities, invest in automated verification, and reserve experienced judgment for the decisions with the greatest consequence.[1]

Education faces the same challenge. Students may conclude that foundations are less important because AI can generate code. The opposite is true. When a system produces an answer quickly, the engineer must recognize whether it is wrong, unsafe, unnecessarily complex, inconsistent with the architecture, or

impossible to operate. Data structures, algorithms, concurrency, state, interfaces, security, testing, and systems thinking become the means by which the engineer challenges the machine rather than merely accepts it.

Professional responsibility

AI can generate an artifact. It cannot assume professional responsibility for the system. The engineer and the organization remain accountable for the consequences of what they accept, release, and operate.

8. ETIS: An Engineering Synthesis for the AI Era

Engineering Trustworthy Intelligent Systems (ETIS) is not intended to replace software engineering, DevSecOps, SRE, NIST, ISO, OWASP, or modern platform practice. Its purpose is to connect them. The industry has accumulated strong disciplines, but they are often applied in isolation: requirements without operational evidence, governance without delivery integration, security without agent authority modeling, CI/CD without decision traceability, AI principles without runtime controls, and observability without stewardship.

ETIS organizes these concerns into a repository-centered lifecycle in which intent, decisions, change, review, verification, release, operation, governance, and learning remain connected. It treats governance as architecture, context as control, and evidence as a first-class engineering product. Its central doctrines are deliberately simple: the model is not the system; AI proposes and engineers verify; everything important leaves evidence; a demo is not operational proof; and trust must be earned across the lifecycle.

This synthesis is especially relevant as engineering agents become participants in delivery. The repository provides their context and constrains their work. Engineering stages provide bounded objectives and phase gates. Evidence provides a basis for human judgment. Operational controls preserve the ability to intervene. Stewardship ensures that responsibility continues after the initial release. ETIS therefore addresses not only how intelligent systems are built, but how organizations preserve control as those systems and their engineering agents gain capability.

The distinct contribution is connective tissue. NIST defines risk outcomes but does not prescribe a GitHub workflow. ISO defines management systems and impact assessment but not the detailed engineering record. DORA explains capabilities and performance but not AI authority. OWASP identifies threats but not the complete lifecycle. Product vendors implement useful platforms but naturally center their own ecosystems. ETIS provides an implementation-neutral engineering model that shows how these concerns meet in day-to-day work.

Professional alignment

ETIS concern	Representative external alignment	How ETIS operationalizes it
Requirements, constraints, and impact	ISO/IEC/IEEE 29148; NIST AI RMF; ISO/IEC 42005	Repository requirements, assumptions, authority boundaries, impact and risk evidence.
Secure and governed development	NIST SSDF; ISO/IEC 42001; ISO/IEC 23894	Integrated design, delivery, and runtime governance with accountable owners.

Agentic security and authority	OWASP Agentic Top 10; least privilege; human oversight	Tool boundaries, permission models, approval gates, monitoring, intervention, and audit.
Delivery performance and reliability	DORA; DevSecOps; Google SRE	Small batches, reviewable change, CI/CD, release evidence, observability, postmortems.
Supply-chain integrity	SLSA; provenance; dependency governance	Traceable build and release baselines, dependency review, verifiable evidence lineage.
Operational understanding	OpenTelemetry; SRE; Well-Architected guidance	Logs, metrics, traces, runbooks, recovery assumptions, incidents, and stewardship.

9. Near-Term Directions

The next several years will not produce a single settled methodology. The market will experiment rapidly, and product terminology will change. Several directions, however, are already visible enough to guide investment and education.

First, specification will become executable context. Requirements, architecture constraints, coding rules, test expectations, security policies, and runbooks will increasingly be consumed directly by engineering agents. Organizations will invest in maintaining those assets because stale context produces stale or unsafe work.

Second, multi-agent engineering will become normal for bounded domains. Specialized agents will analyze requirements, modernize code, generate tests, review changes, investigate incidents, and maintain documentation. Humans will supervise portfolios of agent work rather than interact with one assistant at a time. The quality of orchestration, context partitioning, and review will become a competitive engineering capability.[6, 7, 8]

Third, repositories and delivery platforms will become policy-enforcement environments. Agent permissions, approved tools, protected branches, required checks, provenance, model usage, and review obligations will be configured alongside code. The boundary between engineering platform and governance platform will continue to narrow.

Fourth, assurance will become continuous. Organizations will not be satisfied with a one-time model assessment or release review. They will require ongoing evidence of system behavior, policy compliance, data integrity, security, and realized outcomes. Standards such as ISO/IEC 42001 and 42005, together with NIST and OWASP guidance, will increasingly be translated into automated controls and audit-ready evidence.[12, 13, 15, 16]

Fifth, modernization will be one of the largest practical uses of agentic engineering. Enterprises have decades of valuable but poorly understood software. Agents that can map dependencies, explain code, generate tests, migrate frameworks, and document behavior can reduce the cost of modernization—provided that organizations preserve validation, regression evidence, and operational accountability. IBM Bob's emphasis on enterprise modernization is a strong signal of this market direction.[4, 8]

Finally, the profession will differentiate between organizations that merely license AI tools and organizations that redesign their engineering system. The latter will build healthy data and context ecosystems, clear outcome measures, strong platforms, small-batch workflows, automated verification, and a culture in which humans challenge AI output. The advantage will not come from access to the same models. It will come from the surrounding engineering capability.[1]

10. From AI Adoption to Institutional Capability

Most organizations will begin with tools. Sustainable advantage will come from institutional capability. Tool adoption can be purchased quickly; trustworthy delivery capability must be engineered. It depends on shared practices, platforms, incentives, decision rights, and evidence that survive individual projects and product cycles.

The first institutional requirement is a clear engineering operating model. Teams need to know which decisions remain human, which tasks may be delegated, what evidence is required, how exceptions are handled, and who owns consequences after deployment. Without that clarity, agentic delivery creates a diffusion of responsibility: product believes engineering approved the behavior, engineering believes the model or vendor produced it, and operations inherits a system whose assumptions were never made explicit.

The second requirement is a reusable control plane. Organizations should not force every team to reinvent identity, model access, approved tools, logging, evaluation, secrets handling, dependency policy, and release controls. Platform engineering can provide paved roads that make the responsible path the efficient path. The strongest platforms will combine developer experience with policy enforcement, evidence capture, cost visibility, and operational feedback.

The third requirement is a learning system. AI-assisted delivery is evolving too quickly for governance to be frozen in a policy manual. Teams need controlled experiments, incident analysis, evaluation results, model and tool telemetry, and feedback from users and operators. Those signals should update standards, templates, agent instructions, test suites, and architecture patterns. Continual improvement is not an administrative phase; it is how the organization keeps its controls aligned with changing capability and risk.

The fourth requirement is outcome discipline. Leaders should connect AI investment to customer value, quality, resilience, risk reduction, speed of learning, and employee effectiveness. Measures such as generated code, agent sessions, or completed tickets may describe activity, but they do not prove business value. The more production becomes automated, the more important it is to measure the consequences that the system creates.

Organizations that build these capabilities will use many models and products over time. Their advantage will reside in the engineering system around those products: trusted context, controlled authority, strong platforms, disciplined review, operational evidence, and people able to exercise judgment. That is the durable asset.

Conclusion

Software engineering is not being replaced. It is being elevated. AI lowers the cost of producing code and other artifacts, but it increases the importance of deciding what should be built, how the system should be structured, what authority it should possess, how its behavior will be verified, and how the organization will remain accountable after release.

The decisive shift is from artifact production to trustworthy change. Trustworthy change has a clear purpose, controlled context, coherent architecture, bounded authority, reviewable implementation, credible verification, honest release judgment, observable operation, and accountable stewardship. It leaves evidence that another competent person can inspect.

Executives should therefore treat AI-assisted engineering as an operating-model transformation rather than a tooling rollout. Engineering leaders should strengthen architecture, platforms, context, review, and operational feedback before dramatically increasing change volume. Engineers should learn to use agents aggressively but never surrender understanding or responsibility. Educators should teach students not merely to generate software, but to define, review, govern, and defend it.

Final takeaway

The future software engineer is not simply a faster coder. The future software engineer is the designer, supervisor, validator, governor, and steward of increasingly intelligent systems.

Author Perspective

This paper reflects more than four decades of experience in software engineering, enterprise delivery, architecture, quality, and technology leadership, including service as an IBM Distinguished Engineer and as CTO for IBM Consulting Quality and Delivery. It also reflects current work teaching software engineering at Loyola University Chicago and developing the Engineering Trustworthy Intelligent Systems framework, platform, and educational ecosystem. The perspective is intentionally practical: technology changes quickly, but professional accountability for the systems we release does not.

References

[1] DORA. “State of AI-assisted Software Development 2025.” 2025. https://dora.dev/dora-report-2025/

[2] GitHub. “About GitHub Copilot Cloud Agent.” 2026. https://docs.github.com/copilot/concepts/agents/cloud-agent/about-cloud-agent

[3] OpenAI. “Codex.” 2026. https://openai.com/codex/

[4] IBM. “Introducing IBM Bob: AI Development Partner that Takes Enterprises from AI-Assisted Coding to Production-Ready Software.” 2026. https://newsroom.ibm.com/2026-04-28-introducing-ibm-bob-ai-development-partner-that-takes-enterprises-from-ai-assisted-coding-to-production-ready-software

[5] Amazon Web Services. “AI-Driven Development Life Cycle: Reimagining Software Development.” 2025. https://aws.amazon.com/blogs/devops/ai-driven-development-life-cycle/

[6] GitHub. “About Custom Agents.” 2026. https://docs.github.com/en/copilot/concepts/agents/cloud-agent/about-custom-agents

[7] OpenAI. “Introducing the Codex App.” 2026. https://openai.com/index/introducing-the-codex-app/

[8] IBM. “IBM Advances Enterprise AI Software Development with Multi-Agent Capabilities and Specialized Modernization Workflows.” 2026. https://newsroom.ibm.com/2026-07-09-ibm-advances-enterprise-ai-software-development-with-multi-agent-capabilities-and-specialized-modernization-workflows

[9] OWASP GenAI Security Project. “OWASP Top 10 for Agentic Applications for 2026.” 2025. https://genai.owasp.org/resource/owasp-top-10-for-agentic-applications-for-2026/

[10] MIT Sloan. “Agentic AI, Explained.” 2026. https://mitsloan.mit.edu/ideas-made-to-matter/agentic-ai-explained

[11] OpenAI. “Harness Engineering: Leveraging Codex in an Agent-First World.” 2026. https://openai.com/index/harness-engineering/

[12] National Institute of Standards and Technology. “Artificial Intelligence Risk Management Framework (AI RMF 1.0).” 2023. https://nvlpubs.nist.gov/nistpubs/ai/nist.ai.100-1.pdf

[13] National Institute of Standards and Technology. “AI RMF: Generative Artificial Intelligence Profile.” 2024. https://www.nist.gov/publications/artificial-intelligence-risk-management-framework-generative-artificial-intelligence

[14] International Organization for Standardization. “ISO/IEC 42001:2023—Artificial Intelligence Management Systems.” 2023. https://www.iso.org/standard/42001

[15] International Organization for Standardization. “ISO/IEC 42005:2025—AI System Impact Assessment.” 2025. https://committee.iso.org/committee/6794475/x/catalogue/

[16] OWASP GenAI Security Project. “Agentic AI Threats and Mitigations.” https://genai.owasp.org/resource/agentic-ai-threats-and-mitigations/

[17] OWASP GenAI Security Project. “OWASP GenAI Security Project.” https://genai.owasp.org/

[18] National Institute of Standards and Technology. “Secure Software Development Framework (SSDF).” https://csrc.nist.gov/projects/ssdf

[19] SLSA. “Supply-chain Levels for Software Artifacts.” https://slsa.dev/

[20] ServiceNow. “AI Control Tower.” https://www.servicenow.com/products/ai-control-tower.html

[21] Salesforce. “Agentforce.” https://www.salesforce.com/agentforce/

[22] UiPath. “Agentic AI.” https://www.uipath.com/ai/agentic-ai

[23] International Organization for Standardization. “ISO/IEC 23894:2023—Guidance on AI Risk Management.” https://www.iso.org/standard/77304.html

[24] OpenTelemetry. “Documentation.” https://opentelemetry.io/docs/

[25] Google. “Site Reliability Engineering.” https://sre.google/sre-book/table-of-contents/

[26] ISO/IEC/IEEE. “29148:2018—Requirements Engineering.” https://standards.ieee.org/standard/29148-2018.html

[27] D. L. Parnas. “On the Criteria to Be Used in Decomposing Systems into Modules.” Communications of the ACM, 1972. https://dl.acm.org/doi/10.1145/361598.361623

[28] F. P. Brooks Jr. “No Silver Bullet: Essence and Accidents of Software Engineering.” Computer, 1987. https://www.cs.unc.edu/techreports/86-020.pdf

[29] W. S. Humphrey. Introduction to the Team Software Process. Addison-Wesley, 2000.

[30] GitHub. “Responsible Use of GitHub Copilot Agents.” 2026. https://docs.github.com/en/copilot/responsible-use/agents