ETIS EXECUTIVE BRIEF SERIES

Preparing Engineering Organizations for Agentic Development

A Practical Executive Agenda for Bounded Autonomy, Platform Control, and Operational Trust

William T. O'Connell, Ph.D.

Former IBM Distinguished Engineer and CTO, IBM Consulting Quality and Delivery

Creator, Engineering Trustworthy Intelligent Systems (ETIS)

ETIS Executive Brief EB-003 | Version 1.0 | July 2026

Executive Thesis

Agentic development is not a tool rollout. It is a change in how engineering work is authorized, decomposed, executed, reviewed, and operated. Organizations should expand agent authority only as identity, context, platform controls, evidence, intervention, and accountable ownership become strong enough for the consequence.

Audience: CTOs, CIOs, heads of engineering, chief architects, platform leaders, security executives, enterprise risk leaders, and board technology committees.

Executive Summary

Software agents are moving from assistance to execution. Coding agents can accept assignments, inspect repositories, modify multiple files, run tests, and return pull requests. New orchestration environments allow engineers to launch and supervise multiple agents across repositories and projects [2][3][5]. OpenAI describes agents handling complex, long-running tasks end to end, while GitHub has introduced mission-control capabilities for assigning, steering, and tracking agent work [3][5].

This transition creates substantial opportunity. Organizations can automate routine maintenance, reduce backlog, investigate alternatives in parallel, modernize legacy code, improve test coverage, and extend scarce engineering expertise. But the same capabilities create new failure modes. Agents can misinterpret goals, misuse tools, overreach permissions, consume poisoned context, preserve corrupted memory, propagate errors across connected systems, and produce explanations that encourage unjustified human trust [6][7].

The central executive risk is premature autonomy. Many organizations will scale agent use faster than they scale the engineering system needed to govern it. Local pilots may appear successful because humans compensate informally, the scope is narrow, and consequences are limited. When the same patterns expand across repositories, data, infrastructure, and production environments, hidden assumptions become enterprise risk.

Agentic readiness therefore requires a disciplined operating model. Leaders must define authority levels, establish non-human identity and least privilege, make context authoritative and versioned, preserve provenance, require evidence proportionate to consequence, provide human intervention and rollback, and connect runtime outcomes back to engineering governance. The objective is not to slow adoption. It is to prevent speed from outrunning control.

The Board-Level Question

Where are software agents currently permitted to act, what business consequence can result, and can the organization reconstruct, explain, contain, and reverse their actions?

1. Agentic Development Changes the Unit of Engineering Work

AI-assisted development began with suggestions: code completion, explanation, draft tests, and conversational support. Agentic development changes the unit of work from a suggestion to a delegated objective. The agent determines intermediate steps, selects tools, invokes commands, changes artifacts, and evaluates partial results before returning an outcome.

This distinction matters because control moves upstream. The most consequential decision is no longer only whether a human accepts a generated line of code. It is whether the organization has authorized the agent to choose and perform the actions leading to the result.

Delegation also expands scope. A single agent may touch application code, tests, dependencies, configuration, infrastructure, prompts, data pipelines, and documentation. Multiple agents may work in parallel. The engineering system must therefore govern chains of action rather than isolated outputs.

Mode	Typical agent behavior	Primary executive concern
Assist	Suggests, explains, drafts, or critiques.	Accuracy, disclosure, and human understanding.
Coordinate	Decomposes work, plans tasks, routes	Intent fidelity, context, task boundaries,

	activity, summarizes status.	and handoff quality.
Execute	Changes repositories, runs tools, creates tests, opens pull requests.	Identity, permissions, provenance, review, and rollback.
Operate	Acts in live systems, manages incidents, changes production state, or triggers transactions.	Runtime policy, containment, intervention, accountability, and recovery.

2. Readiness Begins with an Authority Model

Organizations should not discuss agent autonomy as a binary choice. Authority should be expressed as a bounded envelope: permitted goals, repositories, data, tools, environments, action types, transaction limits, time windows, escalation conditions, and prohibited actions.

The authority envelope should be risk-proportionate. An agent drafting unit tests can operate under ordinary review. An agent modifying authentication, financial logic, safety controls, production infrastructure, or regulated data requires stronger separation of duties and independent challenge. A runtime agent needs continuous policy enforcement and immediate revocation.

Authority should also be progressive. New agents begin in sandboxed, recommendation, or simulation modes. They advance only when evidence demonstrates acceptable behavior under increasingly realistic conditions. Authority can expand by task class, system, environment, or consequence. It should contract when incidents, drift, unexplained behavior, or changing context weaken the assurance basis.

Authority Principle

Do not ask whether an agent is trustworthy in the abstract. Ask which actions it is authorized to take, under which conditions, based on which evidence, and with what means of immediate containment.

3. Non-Human Identity Must Become a First-Class Control

An agent that acts through a human's credentials obscures accountability and expands blast radius. Agentic development requires distinct non-human identities that can be authenticated, authorized, monitored, and revoked.

Least privilege should apply to repositories, branches, tools, secrets, data, networks, environments, and deployment targets. Temporary credentials and task-scoped tokens are preferable to standing access. Protected components should require accountable owners. High-consequence actions should require a second identity or human approval.

Identity is also necessary for evidence. The engineering record should distinguish who requested the task, which agent executed it, which tools and models were used, which credentials authorized action, and which human accepted the result. Without that separation, incident reconstruction becomes guesswork.

4. Context Is Part of the Control Plane

Agents act on context: requirements, repository instructions, architecture, examples, policies, test commands, retrieved documents, memory, and conversation history. Poor context produces poor decisions, even when the underlying model is capable.

Context must therefore be treated as a governed engineering asset. Authoritative sources should be identified. Instructions should be versioned and reviewed. Retrieval sources should have provenance and freshness. Sensitive content should be filtered. Memory should have retention, correction, and deletion rules.

OWASP identifies memory and context poisoning as a distinct agentic risk because compromised state can alter behavior long after the initiating interaction [6][7]. The organization needs controls not only over what the agent can do, but also over what it is permitted to believe.

Context layer	Control objective	Representative evidence
Task intent	Ensure the objective is explicit, bounded, and authorized.	Issue, acceptance criteria, constraints, accountable owner.
Repository context	Provide current architecture, instructions, patterns, and tests.	Versioned guidance, ADRs, code ownership, executable commands.
External retrieval	Control provenance, freshness, and trust level.	Source metadata, filters, retrieval logs, approval status.
Memory and state	Prevent persistent corruption and uncontrolled retention.	Memory provenance, retention policy, correction history, access control.
Runtime context	Reflect current policy, identity, environment, and system state.	Policy decisions, tool permissions, environment metadata, telemetry.

5. The Engineering Platform Must Become an Agent Control Plane

Agentic adoption will not scale safely through local scripts and individual subscriptions. The enterprise needs a platform layer that provides approved models, agent identities, tool brokerage, sandboxing, repository controls, policy enforcement, evaluation, provenance, telemetry, secrets handling, and cost visibility.

GitHub's evolution toward agent orchestration and mission control illustrates the direction: agents are becoming managed participants in the repository workflow rather than isolated chat tools [2][3][5]. OpenAI similarly describes agent-first engineering built around repositories, CI, tests, instructions, and parallel task execution [4][8].

The platform should make the governed path the easiest path. Teams should not have to assemble identity, logging, evaluation, and containment independently. Central capabilities should be productized, while product teams remain responsible for the system outcome.

Platform Requirement

An agent platform is not complete when it can invoke a model. It is complete when the organization can define authority, supply trusted context, observe every consequential action, evaluate results, intervene, recover, and preserve evidence.

6. Verification Must Expand from Output Testing to Behavioral Assurance

Traditional software verification remains necessary, but agentic systems require more. The same objective may produce different action paths. The agent may choose different tools, sequence work differently, or encounter unexpected states. Assurance must evaluate behavior, policy adherence, escalation, refusal, side effects, and recovery.

Scenario evaluation should include normal tasks, ambiguous goals, adversarial input, tool failures, stale context, permission denial, partial completion, repeated attempts, and cross-agent interaction. Results should be evaluated as distributions and failure patterns rather than a single demonstration.

Generated tests and self-evaluation are insufficient by themselves. Independent checks, deterministic controls, human review, and production monitoring remain necessary. The stronger the authority, the stronger the independence of assurance.

7. Runtime Governance Is Not Optional

Agents that operate in live environments create a continuing governance obligation. Pre-release review cannot anticipate every state, dependency, adversarial input, or interaction. Runtime control must determine whether the agent remains inside its authority envelope.

Organizations need telemetry for objectives, tool calls, permission decisions, state changes, handoffs, refusals, escalations, human interventions, and outcomes. Alerts should focus on policy violation, unusual action sequences, expanding scope, repeated failure, unexpected cost, and divergence from accepted behavior.

Intervention must be tested. Kill switches, credential revocation, quarantine, rollback, degraded modes, and human takeover should be operational capabilities rather than policy statements. A runtime agent that cannot be stopped safely should not receive consequential authority.

8. Multi-Agent Systems Create Correlated Risk

Multi-agent systems can divide work, apply specialized expertise, and increase throughput. They also create coordination and cascading-failure risks. An incorrect assumption can be repeated by several agents. A compromised agent can misdirect peers. Insecure inter-agent messages can alter goals or authority. Parallel work can create conflicting changes and overwhelm review capacity.

OWASP's agentic guidance highlights insecure inter-agent communication, cascading failures, and human-agent trust exploitation [6][7]. These risks are organizational as well as technical. When agents produce polished explanations and high volumes of work, humans may approve outcomes they have not adequately challenged.

Multi-agent deployment requires explicit orchestration, shared state controls, task ownership, conflict resolution, evidence aggregation, and limits on how one agent may authorize another. Separation of duties should prevent the same agent from proposing, validating, and approving a consequential change.

9. The Human Operating Model Must Be Redesigned

Human oversight is meaningful only when the reviewer has the competence, time, information, authority, and intervention path needed to change the result. Adding an approval button to a fast-moving agent workflow does not create accountability if the human cannot reconstruct the work.

Organizations should define accountable roles for agent owners, platform owners, system owners, model and tool stewards, security, operations, and risk acceptance. Product teams own outcomes; central functions provide shared controls and independent challenge.

Managers must protect review capacity. If agent production grows faster than human understanding, the organization accumulates unreviewed change and false confidence. Small batches, constrained scope, and explicit evidence packages become executive productivity controls, not administrative preferences.

Accountability role	Primary responsibility
Business or product owner	Defines intended value, users, consequence, and accepted business risk.
System owner	Owns architecture, integration, quality, and lifecycle fitness.
Agent owner	Defines agent purpose, authority, evaluation, limitations, and

	retirement criteria.
Platform owner	Provides identity, tools, controls, telemetry, and evidence services.
Security and risk	Challenges threats, control design, exceptions, and residual risk.
Operations	Owns monitoring, intervention, recovery, incidents, and sustained service.
Executive risk authority	Accepts high-consequence residual risk and funds required controls.

10. Common Executive Failure Modes

The first failure mode is pilot extrapolation: assuming a successful low-risk pilot proves enterprise readiness. Pilots often rely on expert attention, narrow context, manual review, and forgiving consequences that do not scale.

The second is agent sprawl. Teams adopt overlapping tools, models, plugins, and local automations without inventory, ownership, cost control, or common evidence. The result is shadow autonomy.

The third is borrowed authority. Agents act through human identities or broadly privileged service accounts, making action attribution and revocation difficult.

The fourth is governance theater: principles, councils, and approval forms exist, but repository rules, platform controls, identity, runtime policy, and intervention do not.

The fifth is review saturation. Agents produce more change than humans can understand. Queue growth is mistaken for productivity.

The sixth is premature production autonomy. Agents are permitted to act in live systems before recovery, monitoring, exception handling, and accountability have been proven.

11. An Agentic Readiness Maturity Model

Level	Operating condition	Executive posture
1. Experimental	Individual assistants and isolated pilots; limited inventory and policy.	Discover use, prohibit high-consequence authority, establish ownership.
2. Controlled Assistance	Approved tools, disclosures, human review, basic data controls.	Standardize access and evidence; build shared platform capabilities.
3. Bounded Execution	Agents perform repository work under scoped identity, tests, and pull-request review.	Define authority tiers, provenance, protected areas, and rollback.
4. Governed Operation	Selected agents act in live systems with runtime policy, telemetry, and intervention.	Use risk-tiered authorization, independent assurance, and continuous monitoring.
5. Adaptive Autonomy	Authority expands and contracts based on operational evidence and changing context.	Treat autonomy as a continuously governed enterprise capability.

12. A 120-Day Executive Agenda

First, create an enterprise agent inventory. Identify where agents suggest, coordinate, execute, or operate; what systems they access; what identities they use; and what consequence can result.

Second, freeze uncontrolled expansion in high-consequence areas until ownership, identity, evidence, and intervention are explicit. This is not a ban on experimentation; it is a boundary on unmanaged authority.

Third, define authority tiers and minimum controls. Establish which tasks require sandboxing, pull-request review, independent approval, runtime policy, or executive risk acceptance.

Fourth, select and fund an agent control platform. Prioritize identity, tool brokerage, context governance, provenance, evaluation, observability, and revocation.

Fifth, redesign two or three engineering value streams end to end. Start with work that is repetitive, well understood, testable, and reversible. Measure business outcome, review burden, quality, cost, and operational impact.

Sixth, establish an agentic engineering review forum. Use it to resolve cross-cutting architecture, security, governance, and platform issues—not to approve every task.

Finally, create a workforce plan. Train engineers to specify, delegate, review, investigate, and operate agentic workflows. Develop early-career engineers through progressively responsible work rather than removing the learning path.

Conclusion

Agentic development will become a normal part of software engineering. The competitive advantage will not come from being first to deploy the largest number of agents. It will come from being first to build an organization that can delegate consequential work without losing understanding, control, or accountability.

The executive challenge is to synchronize capability and authority. Agents can increase the rate at which organizations produce, analyze, review, and operate software. That value is sustainable only when the surrounding engineering system supplies trusted context, strong identity, proportionate evidence, independent challenge, runtime control, and reliable intervention.

Organizations should move decisively—but not casually. Start with bounded tasks, build the platform and governance system, learn from evidence, and expand authority progressively. Agentic development should be treated as an earned operating capability.

ETIS Executive Position

The path to agentic scale is not unrestricted autonomy. It is bounded delegation, repository-centered evidence, governed context, non-human identity, platform-enforced control, meaningful human oversight, and progressively earned authority.

References and Executive Reading

[1] NIST. Artificial Intelligence Risk Management Framework (AI RMF 1.0) and Generative AI Profile. https://www.nist.gov/itl/ai-risk-management-framework

[2] GitHub. Introducing Agent HQ: Any agent, any way you work, 2025. https://github.blog/news-insights/company-news/welcome-home-agents/

[3] GitHub. How to orchestrate agents using mission control, 2025. https://github.blog/ai-and-ml/github-copilot/how-to-orchestrate-agents-using-mission-control/

[4] OpenAI. Harness engineering: leveraging Codex in an agent-first world, 2026. https://openai.com/index/harness-engineering/

[5] OpenAI. Introducing the Codex app, 2026. https://openai.com/index/introducing-the-codex-app/

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

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

[8] OpenAI. Introducing Codex, 2025. https://openai.com/index/introducing-codex/

[9] William T. O'Connell. Engineering Agentic Software Systems. ETIS White Paper WP-004, 2026. https://etisframework.org

[10] William T. O'Connell. Engineering Digital Colleagues. ETIS White Paper WP-010, 2026. https://etisframework.org

[11] William T. O'Connell. Why AI Changes Software Governance. ETIS Executive Brief EB-001, 2026. https://etisframework.org

[12] William T. O'Connell. The Future Software Engineering Organization. ETIS Executive Brief EB-002, 2026. https://etisframework.org

About the Author

William T. O'Connell, Ph.D., is the creator of Engineering Trustworthy Intelligent Systems (ETIS), an adjunct professor of computer science at Loyola University Chicago, and a former IBM Distinguished Engineer and CTO for IBM Consulting Quality and Delivery. His career spans four decades of software engineering, enterprise architecture, product and platform development, quality, governance, and large-scale client delivery.

His work focuses on how technology organizations can use artificial intelligence to increase capability without surrendering engineering discipline, operational responsibility, governance, evidence, or human accountability.