ETIS WHITE PAPER SERIES

WP-004

Engineering Agentic Software Systems

From AI Assistance to Bounded Autonomy, Governed Execution, and Operational Trust

CORE THESIS

The defining engineering question is no longer whether an agent can act. It is whether the system can constrain, observe, verify, interrupt, and account for that action.

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
Creator, Engineering Trustworthy Intelligent Systems (ETIS)

July 2026
ETIS WP-004 | Version 1.0

Executive Abstract

Software agents are moving from conversational assistance to consequential action. They can inspect repositories, plan multi-step work, invoke tools, modify files, run tests, create pull requests, coordinate specialized agents, modernize applications, and increasingly participate in operational workflows. Product direction from OpenAI, GitHub, IBM, Anthropic, Google, and the broader open-source ecosystem indicates that agentic software engineering will become a normal delivery pattern rather than a niche capability. [1-10]

This shift changes the unit of engineering. A model response can be reviewed as content; an agent action must be governed as behavior. The relevant system includes the model, prompts, context, memory, identities, tools, permissions, policies, environments, orchestration logic, evaluations, telemetry, human decision rights, and recovery mechanisms. Failures can arise not only from incorrect output but from goal hijacking, tool misuse, privilege abuse, poisoned memory, unsafe inter-agent communication, cascading action, or a correct action taken under the wrong authority. NIST and OWASP have therefore begun treating AI agents as a distinct standards and security domain. [11-16]

This paper argues that agentic systems require a disciplined engineering model built around bounded autonomy. Authority should be explicit, least-privileged, task-scoped, time-bounded, observable, reversible where feasible, and connected to accountable human ownership. Context should be treated as controlled configuration; tools as privileged interfaces; memory as governed state; agent-to-agent communication as a trust boundary; evaluations as behavioral evidence; and runtime intervention as a first-class architectural capability.

Engineering Trustworthy Intelligent Systems (ETIS) operationalizes these ideas through repository-centered engineering, evidence-centered engineering, design-time and runtime governance, explicit authority boundaries, continuous verification, and operational stewardship. The objective is not to eliminate autonomy. It is to earn it progressively, expanding delegated authority only when context, controls, evidence, and organizational accountability are strong enough for the consequences.

KEY IDEA

Agentic capability is a transfer of authority. Every transfer of authority creates a corresponding obligation for control, evidence, and accountability.

1. The Transition from Assistance to Agency

The first generation of widely adopted generative AI tools operated primarily inside a human-controlled interaction. A developer asked a question, reviewed a suggestion, and decided whether to use it. That model remains valuable, but it is no longer the frontier. Contemporary coding agents can accept issues, work asynchronously, navigate repositories, edit multiple files, execute commands, run tests and security checks, and return a pull request for review. Multi-agent products support parallel tasks, specialized roles, and persistent project instructions. [1-8]

The difference is not merely greater capability. It is delegated execution. An assistant produces an answer; an agent pursues an objective through a sequence of actions. Those actions may alter source code, dependencies, infrastructure, data, configurations, tickets, documents, or live workflows. As action becomes more consequential, the engineering problem moves from output quality to authority management.

This is why the language of 'autonomy' can be misleading. Autonomy is not a single product feature. It is the amount, scope, duration, and consequence of authority transferred from people and deterministic systems to an adaptive actor. A system that drafts a test and waits is materially different from one that changes production configuration, communicates with other agents, or executes a business transaction. Treating all of these as the same category prevents rational control design.

The near-term future is therefore not a binary choice between human work and autonomous work. It is a portfolio of human-agent arrangements. Some agents will assist; others will coordinate; others will execute bounded engineering tasks; a smaller subset will act in operational environments. Engineering leaders will need to classify these arrangements, define decision rights, and establish evidence proportional to consequence.

AUTHORITY MODEL

The progression from assist to operate is not a maturity score. It is an authority gradient.

Level	Agent role	Human control	Primary engineering obligation
Assist	Suggests, explains, drafts, or generates within a human-directed task.	Human reviews every meaningful output before use.	Quality, explainability, and verification.
Coordinate	Plans, sequences, routes, or organizes work across people, tools, and agents.	Human authorizes the plan and critical transitions.	Explicit dependencies, acceptance conditions, and escalation.
Execute	Performs bounded multi-step work using repositories, tools, and test environments.	Human approves policy-defined checkpoints and consequential changes.	Identity, least privilege, sandboxing, provenance, evaluation, rollback.
Operate	Acts in or on behalf of a live system or business process.	Human oversight is risk-based and may be supervisory rather than per-action.	Runtime policy, observability, intervention, containment, accountable ownership.

2. The Agentic System Is Larger Than the Model

A trustworthy agentic system cannot be understood by examining the model alone. The model is one reasoning component within a larger sociotechnical system. Its behavior depends on instructions, retrieved context, memory, available tools, authentication state, permissions, orchestration, environment, human approvals, and feedback from previous actions. A strong model in a weak harness can be unreliable; a capable agent with excessive authority can be dangerous; a well-designed workflow with stale context can execute the wrong intent efficiently.

The practical architecture therefore has several interacting control surfaces. The intent surface defines objectives, constraints, prohibited outcomes, and acceptance evidence. The context surface determines what the agent can know and which sources are authoritative. The authority surface controls which tools, data, systems, and actions are available. The execution surface governs planning, sequencing, retries, parallelism, and inter-agent delegation. The assurance surface evaluates behavior before and during operation. The accountability surface preserves identity, provenance, approvals, exceptions, and ownership.

These surfaces should be designed explicitly. When they remain implicit, control migrates into prompts, undocumented conventions, vendor defaults, or the tacit judgment of individual developers. That may be adequate for experimentation; it is not adequate for consequential systems.

Control surface	Engineering question	Representative evidence
Intent	What outcome is authorized, and what is prohibited?	Requirements, impact record, acceptance criteria, policy constraints.

Control surface	Engineering question	Representative evidence
Context	What may the agent know, trust, and inherit?	Source inventory, context instructions, freshness rules, retrieval tests.
Authority	What may the agent access or change?	Agent identity, tool allowlists, permission scopes, approval gates.
Execution	How may the agent plan, delegate, retry, and stop?	Workflow definition, task graph, limits, timeout and exception policy.
Assurance	What evidence supports safe and correct behavior?	Scenario evaluations, policy tests, regression results, adversarial tests.
Accountability	Who owns the action and its consequences?	Session logs, approvals, provenance, incident ownership, stewardship record.

3. Context Engineering Becomes a Core Discipline

Agent performance depends on context, but more context is not necessarily better context. Context windows are finite; repositories are noisy; enterprise knowledge is fragmented; and instructions can conflict. Anthropic describes context as a critical but finite resource, while GitHub and OpenAI increasingly support repository-level instructions, skills, and project-specific agent behavior. [3, 5, 9, 17]

Context engineering is the disciplined design, selection, packaging, validation, and maintenance of the information an agent needs to act well. It includes requirements, architecture, coding standards, domain terminology, examples, test commands, operational constraints, policies, tool descriptions, and known limitations. It also includes decisions about what to exclude. Irrelevant or stale context can degrade reasoning as surely as missing context.

In agentic systems, context is a control mechanism. It shapes decisions before runtime policy is invoked. A repository instruction that identifies protected modules, required tests, or prohibited dependencies is not merely documentation; it is executable governance. The same is true for an agent skill that encodes how to review a database migration or how to investigate an incident.

Because context controls behavior, it should be versioned, owned, reviewed, tested, and retired. Teams should know which context sources are authoritative, who may modify them, how freshness is assessed, and which evaluations detect harmful drift. Context changes that materially alter agent behavior should be treated as release changes.

ETIS DOCTRINE

Context is not background information. In an agentic system, context is part of the control plane.

4. Tools, Identity, and Authority Boundaries

An agent becomes operationally consequential when it can invoke tools. Tool use converts language into action: reading a database, creating a ticket, merging code, changing infrastructure, sending a message, or initiating a transaction. Model Context Protocol (MCP) is accelerating standardized connections between agents and external data and tools, while NIST is developing guidance specifically for software and AI agent identity and authorization. [18-21]

This interoperability is strategically important, but it expands the attack and failure surface. A convenient tool catalog can become ambient privilege. Tool descriptions may be misleading, external systems may return untrusted content, credentials may be over-scoped, and an agent may combine individually permitted actions

into an unauthorized outcome. The correct security abstraction is therefore not 'the model has access.' It is 'a named agent identity has bounded authority under a policy for a task, in an environment, for a limited time.'

Least privilege must be implemented at the action layer. Read access should be separated from write access; recommendation from execution; staging from production; reversible actions from irreversible ones. High-consequence operations should require stronger authentication, independent approval, or deterministic policy checks. Credentials should be short-lived and attributable. Agents should not inherit the full authority of the human who invoked them merely because that is convenient.

Authorization also needs semantic awareness. A rule that allows an agent to update a customer record may still need to prohibit altering regulated fields, changing records outside the assigned case, or combining data across tenants. Traditional role-based access control remains necessary, but agentic systems increasingly require task-, resource-, and purpose-aware policy.

5. Multi-Agent Systems Create New Trust Boundaries

Multi-agent systems can decompose complex work across specialized roles: planner, implementer, tester, reviewer, security analyst, or operator. OpenAI, GitHub, IBM, and Google are all advancing parallel or specialized-agent models. Agent2Agent (A2A), now under Linux Foundation stewardship, is intended to support secure interoperability among agents across vendors and enterprise platforms. [2, 4, 6-8, 22-24]

Specialization can improve focus and throughput, but it does not automatically create independence. If several agents share the same model, context, assumptions, or faulty source, apparent consensus can be correlated error. An agent that reviews work produced by a sibling agent may reproduce the same blind spots. Multi-agent design should therefore distinguish orchestration from assurance.

Inter-agent communication is itself a trust boundary. Messages can contain untrusted instructions, fabricated claims, sensitive data, or attempts to redirect goals. Agent identities, capabilities, provenance, and conversation state must be visible. A receiving agent should not assume that another agent is authoritative merely because it participates in the workflow.

Cascading failure is a defining risk. One compromised or mistaken agent can produce artifacts that become context for others, triggering a chain of plausible but unsafe actions. Systems need limits on delegation depth, fan-out, retries, spending, data movement, and cumulative consequence. They also need circuit breakers that can halt the workflow before local errors become enterprise actions.

DESIGN WARNING

More agents do not automatically create more assurance. Independent evidence requires independent failure modes.

6. Memory Is Governed State, Not Conversation History

Memory allows agents to preserve preferences, facts, plans, and prior outcomes across tasks. It is essential for continuity, but it also creates durable influence. A poisoned memory can redirect future behavior long after the original interaction is forgotten. An inaccurate summary can become institutional fact. Personal or regulated information can be retained beyond its authorized purpose.

Agent memory should therefore be treated like any other governed data store. Engineers should define what may be remembered, at what scope, for how long, with what provenance, and under whose authority. Users

and operators should be able to inspect, correct, quarantine, or delete memory when appropriate. Sensitive memory should be isolated by tenant, role, purpose, and environment.

The system should distinguish observation from verified fact, recommendation from decision, temporary working state from durable organizational knowledge, and agent inference from human-approved record. Confidence labels alone are insufficient; provenance and validation status matter. Memory writes that affect consequential future action should be reviewable and, where practical, reversible.

This is another area where repository-centered and evidence-centered engineering add value. Durable agent instructions, approved knowledge, decisions, and operational lessons should live in authoritative, governed sources rather than opaque conversational memory whenever possible.

7. Verification Must Evaluate Behavior, Not Only Output

Traditional software testing asks whether specified inputs produce expected outputs. Agentic systems also require evaluation of planning, tool selection, permission use, escalation, stopping behavior, communication, memory updates, and recovery. The same final output may be acceptable or unacceptable depending on how it was produced and what authority was exercised.

Behavioral evaluation should include normal scenarios, ambiguous requests, adversarial inputs, conflicting instructions, unavailable tools, partial failure, stale context, malicious inter-agent messages, and attempts to exceed authority. Evaluations should measure not only task success but policy adherence, unnecessary action, unsafe side effects, escalation quality, reproducibility, cost, and time to intervention.

NIST's work on agent hijacking and agent security, OWASP's Top 10 for Agentic Applications, and the NIST Generative AI Profile all point toward continuous, risk-based evaluation rather than one-time model testing. [11-16, 25] The system should be evaluated at the level at which harm can occur: model, prompt, tool, workflow, identity, environment, and human-agent interaction.

The evaluation set itself is an engineering asset. It should be versioned, linked to risks and requirements, protected from leakage, and updated after incidents. Passing an evaluation is evidence for a bounded claim, not proof of general safety.

8. Human Oversight Must Be Designed, Not Assumed

Many agentic-system descriptions rely on the phrase 'human in the loop.' The phrase is reassuring but incomplete. A person may be nominally present yet lack the context, time, authority, or expertise to intervene effectively. Oversight fails when approvals become routine clicks, alerts are too frequent, agent reasoning is opaque, or the cost of stopping the workflow is organizationally unacceptable.

Effective oversight begins with decision rights. Engineers should identify which decisions must remain human, which may be delegated under policy, which require dual control, and which may be monitored after the fact. The design should specify what the reviewer sees, how uncertainty and risk are communicated, how much time is available, and what happens when the reviewer does nothing.

Oversight should be strongest at irreversible or high-consequence boundaries, not uniformly inserted into every step. Low-value approvals create fatigue and encourage rubber-stamping. Automated controls should handle deterministic checks; humans should focus on ambiguity, tradeoffs, exceptions, and consequence. This division preserves human judgment rather than burying it under generated activity.

The system must also support intervention. Operators need the ability to pause an agent, revoke credentials, disable tools, isolate memory, revert changes, contain affected resources, and preserve forensic evidence. Oversight without intervention is observation, not control.

OVERSIGHT PRINCIPLE

Human oversight is credible only when the human can understand the decision, challenge it, and change the outcome.

9. Runtime Governance and AgentOps

Agentic systems blur the line between application logic and operations. Plans are generated at runtime; tools may be selected dynamically; external systems change independently; and the same objective can produce different action paths. Governance therefore cannot end at design review or deployment. It must continue through operation.

Runtime governance combines policy enforcement, identity, telemetry, evaluation, rate and cost limits, anomaly detection, intervention, incident response, and postmortem learning. The emerging practice is often called AgentOps, but the label matters less than the capability: organizations need to know which agents exist, what authority they possess, what they are doing, what evidence they produce, and how to stop or recover from unsafe behavior.

Observability must capture more than model latency and token usage. Useful signals include agent identity, objective, context sources, tool calls, permission decisions, delegated tasks, policy outcomes, human interventions, state changes, cost, and user or business impact. OpenTelemetry's vendor-neutral approach offers a useful foundation, but agentic semantics require additional conventions and careful privacy controls. [26]

Operational records should support both real-time control and later reconstruction. If an incident occurs, investigators should be able to determine what the agent believed, which sources influenced it, which permissions were used, what changed, who approved exceptions, and whether the system behaved within policy.

10. The Engineering Operating Model for Bounded Autonomy

Organizations will not achieve trustworthy agentic systems through product selection alone. They need an operating model that connects strategy, architecture, platform engineering, security, delivery, operations, governance, and business ownership. The operating model should make authority visible and create a repeatable path from experimentation to production.

A practical model begins with use-case classification. Teams identify intended outcomes, affected stakeholders, consequence, reversibility, data sensitivity, external commitments, and required human authority. The classification determines the permitted autonomy level, approved models and tools, evaluation depth, deployment environment, monitoring, and review body.

Platform teams should provide reusable control planes rather than forcing every product team to rebuild identity, secrets, tool gateways, logging, policy, evaluation harnesses, memory controls, and kill switches. Golden paths accelerate adoption precisely because they embed constraints and evidence. Exceptions should remain possible, but they should be explicit, time-bounded, and owned.

Product and business owners remain accountable for purpose and impact. Engineering owns system integrity; security owns threat-informed controls; operations owns runtime readiness and response; governance owns policy and escalation; and leadership owns incentives. Agentic systems fail institutionally when responsibility is diffused across vendors, models, and teams.

Stage	Organizational posture	Required capabilities	Expansion criterion
1. Assisted	Human-directed use inside existing workflows.	Approved tools, disclosure, review, secure data handling.	Consistent verification and low material incident rate.
2. Bounded execution	Agents perform scoped tasks in controlled environments.	Agent identity, sandboxes, tool scopes, provenance, tests, approval gates.	Reliable task outcomes and effective containment.
3. Coordinated agents	Multiple specialized agents collaborate across workflows.	Delegation policy, inter-agent identity, cumulative limits, independent assurance.	Controlled coordination and explainable failure handling.
4. Operational autonomy	Agents act within live business or technical processes.	Runtime policy, continuous evaluation, intervention, rollback, incident command.	Evidence that outcomes, controls, and stewardship remain effective.
5. Institutional capability	Agentic delivery is governed as an enterprise capability.	Portfolio visibility, reusable platforms, standards alignment, maturity reviews.	Sustained value with bounded risk across products and time.

11. Product and Standards Direction

The product market is converging on several patterns. Coding agents are becoming asynchronous, repository-aware, multi-agent, and integrated with review and validation. Enterprise development platforms are expanding from code generation toward planning, modernization, testing, governance, cost visibility, and operations. Open protocols such as MCP and A2A are creating interoperability layers for tools, data, and agent collaboration. [1-10, 18-24]

Standards direction is following the increase in authority. NIST launched an AI Agent Standards Initiative in 2026 focused on secure, interoperable agents and is developing work on agent identity, authorization, hijacking evaluation, and security. OWASP published a dedicated Top 10 for Agentic Applications covering goal hijacking, tool misuse, identity and privilege abuse, memory poisoning, insecure inter-agent communication, cascading failures, and related threats. [11-16, 20]

Broader AI governance standards remain essential. ISO/IEC 42001 addresses AI management systems; ISO/IEC 23894 risk management; ISO/IEC 42005 impact assessment; ISO/IEC 5338 AI system lifecycle processes; and ISO/IEC 38507 governance implications. These standards do not prescribe a complete agentic engineering architecture, but they establish organizational expectations for accountability, risk, lifecycle management, impact, and continual improvement. [27-31]

The direction is clear: agent ecosystems will become more open and interoperable while control obligations become more explicit. Interoperability without identity, policy, provenance, and containment would scale risk as efficiently as it scales capability.

12. The ETIS Perspective

Engineering Trustworthy Intelligent Systems (ETIS) treats agentic systems as engineered systems of delegated authority. The model is not the system. The system includes intent, context, architecture, identities, permissions, tools, policies, agents, workflows, evidence, operations, governance, and people. Trust emerges from how those elements constrain and support one another across the lifecycle.

Several ETIS doctrines become especially important. Context is control. Governance is architecture. AI proposes; engineers verify. Everything important leaves evidence. A demo is not operational proof. These doctrines convert abstract principles into design obligations. Agent instructions are controlled context. Tool permissions are architecture. Session and action provenance are evidence. Behavioral evaluation is verification. Intervention and rollback are operational proof.

ETIS also rejects two simplistic positions. The first is that autonomy should be maximized because agents are productive. The second is that autonomy should be prohibited because agents are imperfect. The engineering

position is progressive delegation: begin with bounded tasks, measure behavior, strengthen controls, preserve human accountability, and expand authority only when the evidence justifies it.

Repository-centered engineering provides the durable record. Requirements define authorized intent; architecture captures trust and authority boundaries; agent configurations and instructions are versioned; issues and pull requests connect work to purpose; evaluations support claims; release records establish the accepted baseline; telemetry and incidents reveal operational reality; and postmortems drive stewardship. The repository does not contain every runtime event, but it connects the authoritative engineering meaning of those events.

Evidence-centered engineering prevents autonomy from becoming invisible. A trustworthy agentic system should make consequential claims examinable: what the agent was authorized to do, what it actually did, why the action was accepted, what uncertainty remained, how the result was monitored, and who owns the consequences.

ETIS POSITION

ETIS position: autonomy is not granted by product capability. It is earned through bounded authority, credible evidence, operational control, and accountable stewardship.

13. Implications for Leaders, Engineers, and Educators

For executives, the strategic question is not how many agents the organization deploys. It is where delegated authority creates measurable value and whether the organization can control the resulting risk. Leaders should require portfolio visibility, clear ownership, shared platforms, outcome measures, and evidence that controls work under operational conditions. Agent count and token volume are activity metrics, not business outcomes.

For engineering leaders, the priority is to engineer the harness around the model. Invest in specifications, modular architecture, authoritative context, identity, tool gateways, evaluation, observability, and small-batch workflows. Measure review burden and instability as well as generation speed. The bottleneck is already shifting from producing change to understanding and validating it, a trend reflected in current enterprise product direction and industry reporting. [6-8]

For engineers, professional craft moves upward in abstraction without abandoning fundamentals. Engineers must define intent, shape context, design authority boundaries, build test oracles, critique plans, investigate evidence, and operate systems. They also need to understand the implementation sufficiently to recognize when generated work violates architecture, security, or domain meaning.

For security and governance teams, agentic systems require collaboration with platform and product engineering. Policies must be implementable in workflows and runtime controls. Threat models should include goal manipulation, tool composition, memory, inter-agent communication, identity, and cumulative action. Governance that exists only in policy documents will not control dynamic systems.

For educators, agentic engineering makes software engineering education more important. Students should not be evaluated only on code or polished artifacts. They should demonstrate requirements, architecture, traceability, AI disclosure, review quality, behavioral testing, authority reasoning, release readiness, and operational evidence. They should learn to supervise and challenge agents while remaining capable engineers themselves.

Conclusion: Engineer the Authority, Not Just the Intelligence

Agentic AI is changing software engineering because it changes who or what can take action. The economic attraction is clear: agents can work continuously, parallelize tasks, traverse large repositories, connect tools, and perform multi-step work. The engineering challenge is equally clear: action without bounded authority, context integrity, behavioral evidence, and intervention can convert local model error into system consequence.

The profession should therefore resist both automation theater and fear-driven paralysis. Agentic systems can deliver substantial value, especially in modernization, testing, maintenance, operations, and complex workflow coordination. But durable value will come from organizations that treat autonomy as an engineering decision rather than a product setting.

The future software system will increasingly include humans and agents working through shared repositories, platforms, tools, and policies. Its quality will depend less on the eloquence of any single model response and more on the coherence of the surrounding engineering system. Requirements must express authority and prohibited outcomes. Architecture must expose trust boundaries. Context must be governed. Tools must be least-privileged. Memory must be controlled. Agents must be identifiable. Evaluations must test behavior. Operations must support intervention. Evidence must connect every consequential claim to accountable ownership.

That is the central lesson of bounded autonomy: intelligence may propose and act, but engineering must define the conditions under which action deserves trust.

References

[1] OpenAI. “Codex.” 2026. Available online

[2] OpenAI. “Introducing the Codex App.” February 2, 2026. Available online

[3] OpenAI. “Harness Engineering: Leveraging Codex in an Agent-First World.” February 11, 2026. Available online

[4] OpenAI. “Unrolling the Codex Agent Loop.” January 23, 2026. Available online

[5] GitHub. “What’s New with GitHub Copilot Coding Agent.” February 26, 2026. Available online

[6] GitHub. “GitHub Agentic Workflows Is Now in Public Preview.” June 11, 2026. Available online

[7] IBM. “Introducing IBM Bob: AI Development Partner.” April 28, 2026. Available online

[8] IBM. “IBM Advances Enterprise AI Software Development with Multi-Agent Capabilities.” July 9, 2026. Available online

[9] Anthropic. “Effective Context Engineering for AI Agents.” September 29, 2025. Available online

[10] GitHub. “Copilot Coding Agent Now Supports AGENTS.md Custom Instructions.” August 28, 2025. Available online

[11] NIST. “AI Agent Standards Initiative.” 2026. Available online

[12] NIST. “Technical Blog: Strengthening AI Agent Hijacking Evaluations.” January 17, 2025. Available online

[13] NIST CAISI. “Request for Information About Securing AI Agent Systems.” January 12, 2026. Available online

[14] OWASP GenAI Security Project. “OWASP Top 10 for Agentic Applications for 2026.” December 2025. Available online

[15] OWASP GenAI Security Project. “Agentic AI Threats and Mitigations.” 2025. Available online

[16] NIST. “Artificial Intelligence Risk Management Framework: Generative AI Profile, NIST AI 600-1.” 2024. Available online

[17] GitHub. “Agent-Specific Repository Instructions.” November 12, 2025. Available online

[18] Anthropic. “Introducing the Model Context Protocol.” November 25, 2024. Available online

[19] Anthropic. “Donating MCP and Establishing the Agentic AI Foundation.” December 9, 2025. Available online

[20] NIST NCCoE. “Software and AI Agent Identity and Authorization.” 2026. Available online

[21] Anthropic. “MCP Connector.” Claude Platform Documentation, 2026. Available online

[22] Google. “Announcing the Agent2Agent Protocol (A2A).” 2025. Available online

[23] Google. “Google Cloud Donates A2A to Linux Foundation.” June 23, 2025. Available online

[24] Linux Foundation. “Agent2Agent Project.” 2025. Available online

[25] NIST. “Artificial Intelligence Risk Management Framework (AI RMF 1.0).” 2023. Available online

[26] OpenTelemetry Authors. “OpenTelemetry Documentation.” 2026. Available online

[27] ISO. “Artificial Intelligence Standards and Guidance.” 2026. Available online

[28] ISO/IEC 42001:2023. “Artificial Intelligence Management System.” Available online

[29] ISO/IEC 23894:2023. “Artificial Intelligence — Guidance on Risk Management.” Available online

[30] ISO/IEC 42005:2025. “AI System Impact Assessment.” Available online

[31] ISO/IEC 5338:2023. “AI System Life Cycle Processes.” Available online

[32] O’Connell, William T. “ETIS WP-001: Engineering Trustworthy Software in the AI Era.” ETIS White Paper Series, 2026. Available online

[33] O’Connell, William T. “ETIS WP-002: Repository-Centered Engineering.” ETIS White Paper Series, 2026. Available online

[34] O’Connell, William T. “ETIS WP-003: Engineering Evidence.” ETIS White Paper Series, 2026. Available online

About the Author

William T. O'Connell, Ph.D. is an Adjunct Professor of Computer Science at Loyola University Chicago and the creator of the Engineering Trustworthy Intelligent Systems (ETIS) framework. His professional career spans more than four decades of software engineering, architecture, delivery, quality, and technology leadership, including service as an IBM Distinguished Engineer and CTO for IBM Consulting Quality and Delivery. His current work focuses on repository-centered engineering, engineering evidence, trustworthy intelligent systems, agentic software delivery, and the preparation of students and practicing engineers for increasingly AI-assisted engineering environments.

This white paper is part of the ETIS White Paper Series, which examines the engineering practices required to build, govern, review, operate, and sustain trustworthy software and intelligent systems.

Suggested citation: O'Connell, W. T. (2026). Engineering Agentic Software Systems: From AI Assistance to Bounded Autonomy, Governed Execution, and Operational Trust. ETIS White Paper Series, WP-004, Version 1.0.