ETIS EXECUTIVE BRIEF SERIES

Building an AI Engineering Platform

Why Enterprises Need Shared Context, Controls, Evidence, and Paved Roads Before They Need More AI Tools

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-004 | Version 1.0 | July 2026

Executive Thesis

AI tools create local capability. An AI engineering platform converts that capability into an enterprise operating system: shared context, reusable workflows, bounded authority, continuous evidence, economic visibility, and governed paths from intent to production.

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

Executive Summary

Most enterprises are adopting AI from the edges inward. Individual developers acquire assistants. Teams create prompt libraries and local agents. Business units connect models to repositories, data, and tools. These experiments produce valuable learning, but they also create fragmentation: duplicated integrations, inconsistent security, hidden cost, ungoverned context, uneven evaluation, and incompatible evidence.

The strategic response is not another enterprise tool mandate. It is an AI engineering platform: a product-oriented set of shared capabilities that enables teams to design, build, verify, govern, deploy, observe, and improve AI-assisted and agentic systems without recreating the same foundations in every project.

DORA's 2025 research identifies a quality internal platform, strong version control, AI-accessible internal data, small-batch work, a clear AI stance, user focus, and healthy data ecosystems as capabilities that amplify the value of AI [1][2]. The lesson is direct. AI value does not scale through model access alone. It scales through the surrounding engineering system.

Industry platforms are converging on this model. Microsoft Foundry describes a unified platform for building, optimizing, and governing AI applications and agents at scale [4][5]. GitHub is evolving toward a unified workflow for orchestrating multiple agents while retaining repository controls and human review [3]. Backstage provides a software catalog, templates, documentation, and developer portal framework that can expose governed actions to AI clients through MCP [6][7][8]. These products differ, but they point toward the same architectural direction: shared platform capability becomes the control plane for enterprise AI engineering.

The executive objective is not platform centralization for its own sake. The objective is controlled leverage. Product teams should move faster because identity, approved models, context access, evaluation, CI/CD, policy, observability, and evidence are available as reusable services. Governance should become easier because the platform makes authorized behavior visible and the unsafe path harder to take.

The Investment Question

Is the enterprise funding another set of AI tools, or is it building the shared engineering system required to convert AI usage into repeatable, governable, and operationally trustworthy outcomes?

1. Tool Proliferation Is Not Platform Capability

Enterprises commonly begin AI adoption by purchasing access: code assistants, foundation models, agent builders, vector databases, observability tools, evaluation products, and workflow automation. Each can be useful. Together they do not automatically form a platform.

A platform is defined by the operating experience it creates. Teams should be able to move from an authorized problem to a controlled release through repeatable paths. They should not have to determine independently how to authenticate agents, obtain models, classify data, retrieve context, evaluate behavior, preserve provenance, enforce policy, deploy safely, and instrument production.

Without a platform, local innovation creates enterprise entropy. Teams select different tools for similar needs, duplicate connectors, embed secrets, use incompatible telemetry, invent local AI-use policies, and produce evidence that cannot be compared or governed. The organization gains experiments but not compounding capability.

2. The Platform Must Serve Both Humans and Agents

Traditional internal developer platforms improve the human developer experience through service catalogs, self-service environments, templates, documentation, CI/CD, and operational visibility. The AI engineering platform extends that idea. It must support humans who build and review systems and agents that consume context, invoke tools, produce changes, and participate in workflows.

Humans need discoverability, self-service, safe defaults, understandable controls, and clear ownership. Agents need machine-readable context, typed interfaces, scoped identity, deterministic checks, and observable action. The same platform can serve both when engineering knowledge and control are expressed through repositories, catalogs, APIs, policies, templates, and telemetry rather than private conversation.

Backstage illustrates the human-facing foundation: a centralized software catalog, developer portal, documentation, and software templates [6][9]. Its MCP actions capability illustrates the next step: controlled platform actions can be exposed as discoverable tools to AI clients [7][8]. The implication is broader than one technology. The future platform must become legible and usable to both people and software agents.

Platform consumer	Primary need	Platform response
Developer	Fast, safe delivery with low cognitive load	Self-service templates, environments, pipelines, documentation, and feedback.
Reviewer and architect	Context and evidence sufficient for judgment	Traceability, ADRs, policy results, evaluations, provenance, and release records.
AI agent	Authoritative context and bounded capabilities	Machine-readable instructions, catalogs, typed tools, scoped identity, and tests.
Operator	Understanding, intervention, and recovery	Telemetry, runbooks, release baselines, alerts, rollback, and containment.
Executive and risk owner	Portfolio visibility and accountable control	Inventory, ownership, cost, risk tiers, exceptions, and outcome measures.

3. The Eight Capabilities of an AI Engineering Platform

An enterprise platform should be evaluated by capability rather than vendor count. The required capabilities form an integrated system. Weakness in one layer can undermine the rest.

Capability	Executive purpose	Representative platform services
1. Portfolio and software catalog	Know what exists, who owns it, and where risk resides.	Services, repositories, agents, models, MCP servers, owners, lifecycle status.
2. Identity and authority	Ensure every consequential action is attributable and bounded.	Human and non-human identity, task-scoped credentials, least privilege, approval.
3. Model and tool access	Provide approved capabilities without uncontrolled vendor coupling.	Model gateway, routing, tool registry, quotas, safety and contractual controls.
4. Context and knowledge	Supply authoritative, current, permission-aware information.	Repository context, retrieval, data products, policies, memory, provenance.
5. Delivery and evidence	Turn change into reviewable, reproducible engineering records.	Templates, CI/CD, tests, scans, attestations, pull requests, release packages.
6. Evaluation and assurance	Test behavior, policy, quality, and failure before exposure.	Evaluation harnesses, representative datasets, red team, regression, review.
7. Runtime operations	Observe, control, recover, and learn in production.	Tracing, metrics, logs, policy events, cost, intervention, rollback, incidents.
8. Governance and economics	Align adoption with risk appetite and value.	Inventory, risk tiers, exceptions, consumption, unit cost, business outcomes.

4. Context Is the Platform's Strategic Asset

Foundation models are broadly available. Enterprise differentiation increasingly depends on context: domain knowledge, software architecture, customer and operational data, policy, decisions, examples, history, and the procedures through which the organization performs work.

DORA includes AI-accessible internal data and healthy data ecosystems among the capabilities that amplify AI value [2]. Access, however, is not enough. Context must be authoritative, permission-aware, current, attributable, and appropriate to the task. A retrieval service that returns stale or unauthorized information can make a capable model operationally dangerous.

The platform should distinguish source systems, prepared knowledge, task-specific selection, assembled context, memory, and execution evidence. It should identify owners, classifications, freshness, allowed uses, and fallback behavior when sufficient context is unavailable.

Repository context deserves particular attention. Requirements, architecture decisions, code ownership, tests, policies, runbooks, and known limitations are not secondary documentation. They are the operating memory used by human and agentic engineering workflows.

Context Principle

The enterprise advantage is not maximum context. It is minimum sufficient, authoritative, timely, and permission-appropriate context delivered to the right human or agent at the point of decision.

5. Paved Roads Must Include Evidence, Not Only Automation

Platform teams traditionally create paved roads: recommended patterns that reduce setup and standardize delivery. In AI engineering, a paved road is incomplete if it automates deployment but does not preserve evidence.

A professional path should connect authorized intent, repository structure, architecture, implementation, AI contribution, review, evaluation, release, and runtime behavior. Templates should capture required reasoning. Pipelines should preserve test and policy results. Build systems should generate provenance. Releases should establish controlled baselines. Observability should link production outcomes back to the released configuration.

This evidence fabric reduces the cost of governance. Teams should not assemble assurance packages manually at the end. The platform should generate evidence through normal work and make missing evidence visible before approval.

6. Model Gateways and Tool Registries Are Necessary but Insufficient

A model gateway can centralize provider access, authentication, routing, quotas, logging, and cost. A tool registry can make approved capabilities discoverable. These are critical platform services, but they do not create trustworthy engineering by themselves.

The platform also needs use-case context. Which model or tool is appropriate depends on data sensitivity, latency, cost, reliability, explainability, jurisdiction, and consequence. High-impact workflows may require approved configurations, deterministic fallback, additional evaluation, or human authorization.

Tool descriptions are part of the control surface because agents use them to infer capability and action. Registries should include ownership, version, permissions, data classification, expected side effects, test status, and revocation. MCP can standardize how AI applications discover resources and tools, but the enterprise remains responsible for deciding which servers and actions are trusted [7][8].

7. Evaluation Must Become a Shared Enterprise Service

Teams frequently evaluate AI systems through demonstrations or ad hoc prompt testing. That approach does not scale. The platform should provide reusable evaluation harnesses, versioned datasets, scenario libraries, policy tests, comparison methods, regression tracking, and production feedback.

Evaluation should be multi-layered. Model quality matters, but so do retrieval, context, tool use, authorization, workflow, user interaction, cost, escalation, and operational side effects. Agentic systems require behavioral evaluation: whether the system chose appropriate actions, stayed within authority, stopped when uncertain, and recovered from partial failure.

Shared evaluation capability increases speed because teams reuse infrastructure and methods. It also improves governance because results become comparable, traceable, and connected to risk. Passing an evaluation remains evidence for a bounded claim, not proof of general safety.

8. Platform Governance Should Be Federated

A centralized platform team should not own every product decision. Product and service teams possess the domain knowledge and remain accountable for outcomes. Central teams should provide shared controls, architecture, models, data services, security, observability, and assurance methods.

The operating model should be federated. Enterprise leadership sets strategy, risk appetite, funding, and mandatory controls. The platform team builds reusable products. Product teams own use cases, requirements,

implementation, evidence, and operations. Security, risk, architecture, and legal functions provide specialized policy and independent challenge.

This division prevents two extremes. Pure centralization creates queues and weak local ownership. Pure federation creates duplication and uncontrolled risk. The platform provides the common operating system within which local teams exercise bounded autonomy.

Layer	Accountability	Primary platform relationship
Enterprise leadership	Strategy, risk appetite, investment, portfolio outcomes	Funds platform, sets mandatory boundaries, reviews enterprise measures.
Platform organization	Shared capability, usability, reliability, and controls	Operates platform as a product with customers and service objectives.
Product and service teams	Use-case value, engineering evidence, release, and operations	Consume paved roads; own deviations, risks, and business outcomes.
Architecture, security, risk, legal	Specialized policy, challenge, and exception decisions	Encode reusable controls; review high-consequence exceptions.
Data and knowledge owners	Authority, quality, access, retention, and permissible use	Publish governed data and context products for humans and agents.

9. Treat the Platform as a Product, Not an Infrastructure Program

Many internal platforms fail because they are built as central infrastructure projects rather than products. Teams are told to adopt a portal, pipeline, or standard without evidence that it improves their work. Adoption becomes compliance rather than value.

The AI engineering platform needs explicit customers, product management, service ownership, user research, a roadmap, support, and measurable outcomes. Its success depends on whether teams voluntarily choose the paved road because it is faster, safer, and easier than local assembly.

Measures should include adoption by target workflows, onboarding time, lead time, review burden, evaluation reuse, control coverage, developer satisfaction, platform reliability, unit cost, incident reduction, and business outcomes. Tool logins and token consumption are not platform-value measures.

Platform Product Principle

The platform succeeds when it makes trustworthy engineering easier. A mandated control surface that teams routinely bypass is not a platform; it is a source of shadow infrastructure.

10. Architecture: A Reference Control Plane

A practical reference architecture separates experience, control, execution, and evidence. The experience layer includes the developer portal, repositories, IDEs, APIs, and agent orchestration. The control layer includes identity, policy, catalog, model gateway, tool registry, data classification, and approvals. The execution layer includes CI/CD, sandboxes, agent runtimes, cloud environments, and operational systems. The evidence layer includes evaluations, provenance, telemetry, cost, releases, incidents, and audit records.

These layers should be connected through durable identifiers. A production action should be traceable to an agent identity, assignment, context sources, tool calls, policy decisions, released configuration, and accountable owner. This is what turns a collection of technologies into an engineering system.

Architecture layer	Core components	Control objective
Experience	Portal, repositories, IDEs, APIs, orchestration, self-service	Make the governed path discoverable and efficient.
Control	Catalog, identity, policy, gateway, tool registry, approvals	Define what exists, who may act, and under which conditions.
Execution	Pipelines, sandboxes, agent runtimes, cloud and enterprise systems	Perform bounded work through controlled environments.
Evidence	Evaluations, provenance, telemetry, cost, releases, incidents	Support review, intervention, economics, and learning.

11. Build-versus-Buy Is the Wrong First Question

Executives often begin platform strategy by comparing products. The prior decision is architectural: which capabilities are differentiating, which are control points, which should remain portable, and which can be consumed as managed services.

Most enterprises will assemble a platform from commercial services, cloud capabilities, open-source frameworks, internal software, repositories, and data systems. Microsoft Foundry can provide managed model and agent services [4][5]. GitHub can provide repository-centered workflow and agent orchestration [3]. Backstage can provide catalog and portal capabilities [6][9]. These are examples, not a mandatory stack.

The enterprise should retain control of identity, policy, ownership, evidence, data authority, and portability decisions even when execution services are purchased. Vendor products should fit the operating model; the operating model should not be inherited accidentally from vendor boundaries.

12. A 180-Day Executive Agenda

First, inventory the current landscape. Identify AI tools, models, agents, data connections, repositories, evaluation methods, identities, costs, and owners. The platform strategy must begin with reality rather than the approved-tool list.

Second, define the platform product and target customers. Select a small number of high-value engineering journeys, such as repository coding agents, AI-assisted modernization, or an internal agent application path.

Third, define the minimum control plane: catalog, ownership, non-human identity, model gateway, tool registration, context governance, evaluation, provenance, telemetry, and revocation.

Fourth, build one end-to-end paved road. It should begin with authorized intent and end with a controlled release and operational evidence. Measure lead time, quality, review effort, cost, and user experience.

Fifth, create the federated operating model. Establish platform ownership, product-team accountability, control owners, exception paths, and funding.

Sixth, publish a portability and sourcing strategy. Decide where the enterprise will standardize, where it will support choice, and which data, identity, evidence, and policy layers must remain independent of model vendors.

Finally, scale based on evidence. Expand platform services and agent authority only when adoption, delivery performance, quality, economics, and operational outcomes demonstrate value.

Conclusion

AI engineering will not scale through unlimited access to models and tools. It will scale through an enterprise platform that turns fragmented capability into a repeatable operating system.

The platform's purpose is not to centralize every decision or eliminate local innovation. It is to provide the common context, identity, controls, workflows, evidence, and operational services that allow product teams to move quickly without recreating enterprise risk.

Organizations that build this capability will gain more than developer productivity. They will create a reusable foundation for agentic development, software modernization, AI-enabled products, governance, and continuous organizational learning. The platform becomes the place where AI ambition is converted into trustworthy engineering.

ETIS Executive Position

Before scaling AI tools, build the engineering system that makes their use coherent: repository-centered workflow, shared context, bounded authority, reusable evaluation, continuous evidence, runtime control, and accountable ownership.

References and Executive Reading

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

[2] DORA. AI Capabilities Model and Platform Engineering capability guidance. https://dora.dev/ai/

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

[4] Microsoft. What is Microsoft Foundry? https://learn.microsoft.com/en-us/azure/foundry/what-is-foundry

[5] Microsoft. Microsoft Foundry Agent Service overview. https://learn.microsoft.com/en-us/azure/foundry/agents/overview

[6] Backstage. What is Backstage? https://backstage.io/docs/overview/what-is-backstage/

[7] Backstage. MCP Actions Backend. https://backstage.io/api/next/modules/_backstage_plugin-mcp-actions-backend.html

[8] Backstage. Release v1.40.0: Actions Registry and MCP actions. https://backstage.io/docs/releases/v1.40.0/

[9] Backstage. Software Templates. https://backstage.io/docs/features/software-templates/

[10] William T. O'Connell. Repository-Centered Engineering. ETIS White Paper WP-002, 2026. https://etisframework.org

[11] William T. O'Connell. Context Engineering. ETIS White Paper WP-009, 2026. https://etisframework.org

[12] William T. O'Connell. Preparing Engineering Organizations for Agentic Development. ETIS Executive Brief EB-003, 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.