ETIS WHITE PAPER SERIES

WP-007

Engineering Review and Readiness

Turning Inspection, Challenge, and Evidence into Trustworthy Change

Core Thesis

Engineering review is not a ceremonial pause before approval. It is the disciplined mechanism through which a team challenges claims, exposes hidden risk, improves decisions, and determines whether a change is ready to advance.

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-007 | Version 1.0

Executive Abstract

Software engineering has always depended on review, but review is often treated too narrowly: as a code-reading activity, a required meeting, or an approval step placed near the end of a delivery process. That view is no longer adequate. Modern systems are assembled from code, services, data, configurations, infrastructure, models, prompts, policies, dependencies, and agent permissions. AI-assisted development can create these artifacts faster than organizations can understand them. As production capacity rises, the limiting factor becomes the ability to evaluate change before it becomes operational consequence.

This paper argues that engineering review should be understood as a continuous assurance system. Requirements reviews test whether intent is clear and bounded. Architecture reviews test whether system structure, authority, data, and failure behavior are defensible. Pull-request reviews test whether a proposed change improves the system without degrading its health. Security and AI reviews examine threats, misuse paths, provenance, and human oversight. Release and operational-readiness reviews determine whether evidence is sufficient to expose the change to users and operations. Postmortems review actual outcomes and return learning to the lifecycle.

The paper synthesizes established review practice from IEEE software-review standards, Google engineering guidance, GitHub collaboration workflows, NIST secure-development and AI-risk guidance, OWASP secure-code-review practice, AWS Well-Architected reviews, and Google Site Reliability Engineering. It then develops the ETIS position: review is the engineered conversion of uncertainty into accountable judgment. A trustworthy organization does not merely ask whether work was reviewed. It asks whether the right people challenged the right claims, using the right evidence, at the right time, and whether findings were resolved or explicitly accepted.

Executive Takeaway

AI can accelerate production, but it cannot confer readiness. Readiness is an engineering judgment supported by evidence and strengthened by disciplined challenge.

1. Review as an Engineering Control System

A review is often described as a meeting in which people inspect work. That description captures the visible event but misses the engineering function. The deeper purpose of review is to reduce uncertainty before authority is exercised. A team proposes a requirement, design, code change, release, or operational decision. The review tests the proposal against intent, constraints, risk, and evidence. The result is not simply approval or rejection. A serious review produces clarified assumptions, improved work, explicit findings, accountable owners, and a defensible decision about what happens next.

IEEE 1028 formalized multiple review forms—management reviews, technical reviews, inspections, walkthroughs, and audits—because different decisions require different forms of challenge [1]. That distinction remains useful, but contemporary engineering requires a broader interpretation. The object under review is no longer only source code or a specification. It may include infrastructure definitions, deployment pipelines, third-party dependencies, training or evaluation data, prompts, model configurations, agent tools, identity policies, and observability controls. The review system must expand with the engineered system.

Within ETIS, review is a control system connecting intent, implementation, evidence, and authority. It is the point at which engineering claims are challenged before they become commitments. It is also a knowledge-transfer mechanism: reviewers learn the system, authors explain their reasoning, and the repository captures

the resulting record. Review therefore contributes simultaneously to quality, governance, resilience, and organizational memory.

ETIS Doctrine
A review is credible only when it can change the work, change the decision, or change the conditions under which the work may proceed.

2. From Inspection to Continuous Assurance

Traditional review models were often episodic. A design document was reviewed at a milestone, code was inspected before release, and an audit checked whether prescribed procedures had been followed. Those practices remain valuable, but rapid delivery and AI-assisted production make episodic review insufficient. Changes now arrive continuously, dependencies update automatically, infrastructure is expressed as code, and intelligent components may alter behavior without a conventional source-code deployment.

Continuous assurance does not mean that every decision requires a committee. It means that review is designed into the flow of engineering work. Routine, low-risk changes may be evaluated through automated checks and peer review. High-risk architectural, security, data, AI-authority, or release decisions require broader expertise and stronger evidence. The review burden should be proportional to the authority, impact, irreversibility, and uncertainty of the proposed change.

GitHub pull-request workflows illustrate this shift. A pull request is simultaneously a proposed change, a review surface, a discussion record, a validation context, and a merge decision [2]. Google's public engineering guidance similarly treats code review as a mechanism for improving the long-term health of a codebase, not merely catching immediate defects [3]. NIST's Secure Software Development Framework recommends peer review of code and consideration of existing analysis and test results as part of secure development [4]. Together these practices show that review is becoming persistent, traceable, and integrated with delivery.

The same principle applies beyond code. AWS describes architecture review as a lightweight, blame-free conversation intended to identify critical issues and actionable improvements rather than to conduct an audit [5]. Google SRE uses blameless postmortems to examine operational outcomes and convert failure into systemic learning [6]. The common pattern is review as a feedback system, not a gatekeeping ritual.

3. The Review Continuum Across the Lifecycle

Engineering review should follow the lifecycle because uncertainty changes as work matures. Early reviews ask whether the team understands the problem. Middle-stage reviews ask whether the proposed solution is coherent and safe. Delivery reviews ask whether the implementation is correct, traceable, and supportable. Operational reviews ask whether the system behaves acceptably in reality. No single review can answer all of these questions.

Review Layer	Primary Question	Representative Evidence
Intent and requirements	Does the proposed work solve the right problem, for the right stakeholders, within explicit constraints?	Requirements, acceptance criteria, assumptions, risk and authority boundaries
Planning and commitment	Is the work bounded, owned, sequenced, and feasible?	Scope, estimates, dependencies, risks, milestones or delivery targets
Architecture and design	Are responsibilities, interfaces, data, authority, failure modes, and tradeoffs defensible?	Architecture views, ADRs, threat models, data and context boundaries

Review Layer	Primary Question	Representative Evidence
Implementation and pull request	Does the change improve the system without reducing code health, security, testability, or maintainability?	Issue, diff, tests, analysis, reviewer comments, CI results
AI and agent behavior	Are generated artifacts, model behavior, tool permissions, human oversight, and residual uncertainty acceptable?	AI-use records, evaluations, policy tests, tool and authority maps
Release readiness	Is the complete release claim supported, reversible, observable, and honest about limitations?	Release baseline, test evidence, defects, rollback and runbook evidence
Operations and incident learning	Did the system behave as expected, and what must change based on actual outcomes?	Telemetry, incident records, postmortems, corrective actions

This continuum also prevents a common failure: using a late review to discover problems that should have been challenged earlier. A release review cannot compensate for vague requirements. A code review cannot repair a fundamentally unsound architecture. A postmortem can expose a governance failure, but it cannot undo the harm already caused. Effective review moves important questions as early as the available evidence allows, then revisits them as operational knowledge improves.

4. What Makes a Review Credible

A review is not credible merely because it occurred. Review quality depends on preparation, independence, evidence, competence, scope, and closure. Weak reviews often fail because participants do not know what decision is being made, authors present conclusions without supporting evidence, reviewers lack the necessary perspective, or findings disappear after the meeting.

A credible review begins with an explicit claim. The team may claim that a requirement is ready, an architecture is safe enough to implement, a change is safe enough to merge, or a release is ready for operational exposure. The review then identifies the evidence required to evaluate that claim. This makes review focused and prevents the meeting from becoming a general discussion.

Reviewer competence must match the claim. A user-experience review requires different expertise than a security review. An agent-authority change may require engineering, security, domain, legal, and operational perspectives. Independence also matters, but independence does not always require an external board. It requires enough separation for reviewers to challenge assumptions without merely defending their own work.

The review must end with disposition. Findings should be accepted, rejected with rationale, deferred with explicit risk ownership, or converted into tracked actions. A vague statement that “the team discussed the issue” is not closure. Closure requires a decision, an owner, a due date or condition, and evidence that the response was implemented or the residual risk knowingly accepted.

Review Principle

Review is not complete when comments are written. It is complete when findings are dispositioned and the engineering record shows what changed, what did not, and why.

Credibility Dimension	Weak Pattern	Strong Pattern
Purpose	Review everything generally	State the claim and decision under review

Credibility Dimension	Weak Pattern	Strong Pattern
Preparation	Read material during the meeting	Provide current evidence before the review
Challenge	Seek consensus quickly	Invite reasoned disagreement and alternative explanations
Evidence	Rely on confidence or demonstration	Connect claims to reproducible, inspectable evidence
Disposition	Record comments without closure	Assign findings, owners, rationale, and closure evidence
Learning	Treat issues as individual mistakes	Identify systemic improvements and update standards

5. Pull Requests and the Economics of Small-Batch Review

The pull request has become the most common review unit in repository-centered engineering. At its best, it expresses intent, connects the work to an issue or requirement, exposes a bounded change, presents validation evidence, records discussion, and preserves the decision to merge. At its worst, it is a large diff with an empty description and a ceremonial approval.

Google's review guidance emphasizes that the primary objective is to improve code health over time and that reviewers should consider design, functionality, complexity, tests, naming, comments, style, and documentation [3][7]. It also advocates small changes because they are easier to design, review, test, and roll back [8]. GitHub provides review states, comments, requested changes, code-owner routing, and protected-branch controls that can transform the pull request into a governance surface [2][9].

Small-batch review is not merely a developer preference. It is an economic control. Review effort grows faster than line count because interactions, assumptions, and failure paths multiply. Large changes increase reviewer fatigue, reduce comprehension, delay feedback, and create pressure to approve because the cost of rework is already high. Small changes reduce the amount of simultaneous uncertainty and allow defects or architectural drift to be corrected before they spread.

AI-assisted implementation makes this discipline more important. A coding agent can produce a large, syntactically coherent change in minutes. That does not make the change reviewable. Teams must decompose agent work into bounded units, require meaningful descriptions, demand tests and provenance, and preserve human responsibility for merge decisions. Review speed should increase through better preparation and smaller changes—not by reducing scrutiny.

6. Reviewing AI-Assisted and Agentic Work

AI-assisted engineering changes both the volume and the nature of review. A reviewer may encounter generated code, tests, requirements, architecture summaries, migration scripts, infrastructure configuration, or incident analysis. The artifact may appear polished while containing hidden assumptions, fabricated dependencies, incomplete tests, or design choices inconsistent with the surrounding system. Fluency is not evidence of correctness.

NIST's AI Risk Management Framework emphasizes documentation, human review, accountability, and risk management across the AI lifecycle [10]. Its Generative AI Profile notes that generative systems may warrant additional review, tracking, documentation, and management oversight [11]. These expectations apply not only to deployed AI products but also to AI used inside the engineering process. If AI contributes materially to a change, the review should establish what was generated, what humans modified, how the result was verified, and what uncertainty remains.

Agentic engineering raises the stakes further. An agent may inspect a repository, plan work, change files, run commands, select dependencies, open pull requests, or modify infrastructure. Review must therefore address authority as well as output. Reviewers should know which tools the agent used, which data and repositories it accessed, whether commands were bounded, what tests ran, and whether the agent crossed architectural or policy boundaries. The more authority delegated, the stronger the evidence and independent challenge required.

AI should also be used as a reviewer, but not as the final authority. Automated review can identify patterns, missing tests, suspicious dependencies, style problems, and potential defects. GitHub now supports AI-assisted code review within pull-request workflows [12]. Such systems can increase coverage and reduce repetitive work, but they may miss context-specific risks or produce confident false positives. AI review should supplement—not replace—qualified human judgment.

Review Question	Why It Matters
What did AI generate or materially influence?	Establishes provenance and accountability.
What context and constraints were supplied?	Reveals whether the system was grounded in current engineering intent.
What was accepted, modified, or rejected?	Shows human judgment rather than passive adoption.
How was the output verified?	Connects acceptance to tests, analysis, comparison, or expert review.
What authority did the agent exercise?	Determines the required strength of controls and oversight.
What uncertainty or limitation remains?	Prevents polished output from becoming an overstated claim.

7. Release and Operational Readiness

A release review is qualitatively different from a code review. The object under review is the assembled system and the claim that it is ready for a defined level of exposure. The evidence must therefore include more than passing unit tests. Reviewers need to understand what changed, what remains incomplete, what failures are plausible, how the system will be observed, how operators will respond, and how the release can be contained or reversed.

AWS Well-Architected reviews frame architecture review as a constructive, consistent conversation that produces actionable improvements across operational excellence, security, reliability, performance, cost, and sustainability [5][13]. Google SRE similarly treats launch and operational review as part of engineering, not as a separate production handoff [14]. These models reinforce a central ETIS principle: a demo is not operational proof.

Readiness is contextual. A classroom prototype, internal pilot, regulated decision system, and autonomous production agent do not require identical evidence. The review should define the intended operating boundary and judge readiness against that boundary. A system may be ready for a supervised pilot but not for autonomous production. A release may be acceptable with a known limitation if the limitation is visible, mitigated, and owned. Honest bounded readiness is stronger than an inflated claim of completeness.

The review record should identify the release baseline, evidence considered, unresolved defects, known limitations, residual risks, mitigations, rollback conditions, operational ownership, and the decision. This record turns release approval from an ephemeral meeting into durable engineering evidence.

8. Review Culture, Independence, and Human Judgment

Review is a technical process embedded in a social system. The same checklist can produce very different outcomes depending on culture. In low-trust environments, authors hide uncertainty, reviewers perform status, and findings become personal criticism. In mature environments, uncertainty is expected, challenge is professional, and the purpose is to improve the system rather than assign blame.

Google's engineering guidance recommends courteous, clear comments focused on the code rather than the person [15]. AWS advocates blame-free architecture review [5]. Google SRE's postmortem model emphasizes learning from failure and converting analysis into corrective action [6][16]. These practices do not lower standards. They make rigorous challenge sustainable by separating accountability for improvement from personal blame.

Independence must also be designed. Authors should not be the sole approvers of their own high-impact changes. Specialized risks should be reviewed by qualified specialists. Reviewers should disclose conflicts and avoid rubber-stamping work from senior or influential authors. At the same time, organizations should avoid creating distant review boards that lack system context and merely impose delay. The goal is informed independence: enough context to understand the system and enough separation to challenge it.

Human judgment remains central because review involves tradeoffs, not just rule checking. Automated tools can detect patterns; humans must decide whether the system behavior, architecture, and residual risk are acceptable for the intended context. Meaningful human oversight requires authority, competence, time, and information. A reviewer who cannot stop the change, lacks relevant evidence, or is pressured to approve is not providing meaningful oversight.

9. Failure Modes and Review Theater

Organizations often possess the visible machinery of review while receiving little assurance from it. Review theater occurs when process evidence substitutes for engineering challenge: a meeting is held, a template is completed, or an approval is recorded, but no one tests the underlying claim.

Failure Mode	What It Looks Like	Consequence
Rubber-stamp approval	Approvals arrive without comments or evidence	Defects and risk pass through an appearance of control
Review after commitment	Architecture or scope is reviewed only after implementation	Feedback becomes too expensive to adopt
Oversized change	Thousands of lines, mixed concerns, weak description	Reviewer comprehension collapses
Checklist substitution	Boxes are checked without examining context	Compliance evidence masks engineering weakness
Missing expertise	Reviewers lack security, operations, domain, or AI knowledge	Critical risk dimensions remain unchallenged
Unclosed findings	Comments or action items remain unresolved	Known problems become invisible residual risk
AI deference	Generated output is trusted because it appears polished	Fabricated or context-inconsistent work is accepted
Blame culture	Authors hide uncertainty and reviewers avoid difficult findings	The organization loses learning and early warning

The cure is not more review meetings. It is better review design: clearer claims, smaller review units, stronger evidence, competent reviewers, explicit authority, and visible disposition. Review overhead should be reduced by improving the quality of inputs and automating repetitive checks, not by eliminating challenge.

10. An Engineering Review Operating Model

A mature organization treats review as a portfolio of risk-proportionate practices rather than a single process. The operating model defines which changes require which reviews, who may approve them, what evidence is mandatory, how findings are tracked, and how learning updates engineering standards.

At the team level, routine changes flow through issues, branches, pull requests, peer review, automated checks, and merge controls. At the system level, material changes trigger architecture, security, data, AI, or operational review. At the organizational level, governance bodies define decision rights, risk tiers, exception paths, and escalation. The repository connects these levels by preserving the evidence and decisions.

Operating-Model Element	Design Question
Review taxonomy	What kinds of review exist, and which decisions do they support?
Risk triggers	What characteristics increase review depth: authority, data sensitivity, irreversibility, user impact, novelty?
Decision rights	Who may recommend, approve, request changes, or accept residual risk?
Evidence contract	What evidence must exist before the review begins?
Workflow integration	How are reviews connected to issues, pull requests, checks, releases, and operational records?
Finding disposition	How are actions, exceptions, due dates, and closure evidence tracked?
Metrics and learning	How does the organization measure review quality and improve standards?

Metrics should be chosen carefully. Counting approvals or comments can reward noise. Better indicators include review latency, change size, defect escape rate, rework after review, finding closure time, recurring root causes, reviewer coverage, and the proportion of high-risk changes receiving qualified independent review. Metrics should reveal whether review improves outcomes, not merely whether process activity occurred.

11. A Five-Level Review Maturity Model

Maturity Level	Characteristics
Level 1 — Ad Hoc	Review depends on individual diligence; evidence and decision rights are inconsistent.
Level 2 — Repeatable	Teams use common pull-request and checklist practices, but review scope remains mostly local.
Level 3 — Integrated	Requirements, architecture, security, AI, release, and operational reviews are connected through repository evidence.
Level 4 — Risk-Proportionate	Review depth, independence, and evidence requirements vary

Maturity Level	Characteristics
	explicitly with authority and impact.
Level 5 — Learning System	Review outcomes, operational evidence, and postmortems continuously improve standards, automation, training, and architecture.

Maturity should not be confused with bureaucracy. A Level 5 organization may conduct fewer meetings than a Level 2 organization because expectations are clear, evidence is current, changes are small, and routine controls are automated. The distinguishing feature is not process volume. It is the organization's ability to apply effective challenge where it matters and to learn from the results.

12. The ETIS Position

ETIS treats engineering review as one of the principal mechanisms through which trust is earned. Repository-centered engineering provides the review surface. Evidence-centered engineering provides the basis for judgment. Engineering governance defines decision rights and escalation. Operational evidence tests whether prior judgments remain valid. Review connects all four.

The ETIS position can be stated in six propositions. First, every consequential engineering claim should be reviewable. Second, review depth should increase with authority, impact, uncertainty, and irreversibility. Third, reviewers must be able to inspect the evidence without relying on private explanation. Fourth, AI may propose and assist with review, but accountable humans retain the decision. Fifth, findings require explicit disposition and closure evidence. Sixth, operational outcomes must flow back into requirements, architecture, controls, and future reviews.

ETIS Position
Trustworthy systems are not produced by approval alone. They are produced when challenge improves the work, evidence supports the decision, and learning changes what the organization does next.

Implications for Leaders, Engineers, and Educators

For executives and engineering leaders, the central implication is that review capacity becomes strategic as AI increases production capacity. Organizations that invest only in generation tools will create an assurance bottleneck. They need review architecture: risk tiers, decision rights, reviewer capability, evidence requirements, automated checks, and clear escalation. Review should be treated as part of the delivery system and resourced accordingly.

For engineers, review becomes a core professional capability. Authors must learn to present bounded changes, explain intent and tradeoffs, disclose uncertainty, and produce verification evidence. Reviewers must reason across architecture, security, operations, data, AI behavior, and long-term maintainability—not only syntax. The ability to challenge respectfully and decide responsibly will matter more as routine implementation becomes increasingly automated.

For educators, review should be taught as engineering practice rather than grading ritual. Students should review requirements, architecture, pull requests, test evidence, AI use, release claims, and postmortems. They should learn to distinguish comments from findings, findings from decisions, and decisions from closure. A student who can produce code but cannot review, defend, or revise engineering work is not yet prepared for AI-era professional responsibility.

Conclusion

Software engineering review is often underestimated because its output is less visible than code. Yet review is where organizations convert individual production into collective judgment. It is where assumptions become explicit, risks are surfaced, evidence is tested, and authority is either granted or withheld.

The AI era makes this function more important. Generated work arrives faster, appears more complete, and may include decisions or actions that are difficult to reconstruct. Agentic systems can exercise authority across repositories, tools, data, and operations. In that environment, review cannot remain a late manual inspection. It must become a continuous, risk-proportionate assurance system spanning requirements, architecture, implementation, AI behavior, release, and operations.

The future of review is not more bureaucracy. It is more precise challenge, better evidence, smaller changes, qualified independence, and faster learning. Organizations that engineer review well will move quickly with justified confidence. Those that treat review as ceremony will simply automate the production of unexamined risk.

Final Takeaway

AI changes how fast engineering work can be produced. Review determines whether that work deserves to become part of the system.

References

[1] IEEE Standards Association. IEEE Std 1028-2008, IEEE Standard for Software Reviews and Audits. https://standards.ieee.org/standard/1028-2008.html

[2] GitHub Docs. About pull request reviews. https://docs.github.com/pull-requests/collaborating-with-pull-requests/reviewing-changes-in-pull-requests/about-pull-request-reviews

[3] Google Engineering Practices. The Standard of Code Review. https://google.github.io/eng-practices/review/reviewer/standard.html

[4] National Institute of Standards and Technology. Secure Software Development Framework (SSDF) Version 1.1, NIST SP 800-218. https://csrc.nist.gov/pubs/sp/800/218/final

[5] Amazon Web Services. AWS Well-Architected Framework: The Review Process. https://docs.aws.amazon.com/wellarchitected/latest/framework/the-review-process.html

[6] Google Site Reliability Engineering. Postmortem Culture: Learning from Failure. https://sre.google/sre-book/postmortem-culture/

[7] Google Engineering Practices. What to Look for in a Code Review. https://google.github.io/eng-practices/review/reviewer/looking-for.html

[8] Google Engineering Practices. Small CLs. https://google.github.io/eng-practices/review/developer/small-cls.html

[9] GitHub Docs. Managing pull request reviews in your repository. https://docs.github.com/en/repositories/managing-your-repositorys-settings-and-features/managing-repository-settings/managing-pull-request-reviews-in-your-repository

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

[11] National Institute of Standards and Technology. Artificial Intelligence Risk Management Framework: Generative Artificial Intelligence Profile, NIST AI 600-1. https://nvlpubs.nist.gov/nistpubs/ai/NIST.AI.600-1.pdf

[12] GitHub Docs. About GitHub Copilot code review. https://docs.github.com/en/copilot/concepts/agents/code-review

[13] Amazon Web Services. AWS Well-Architected Framework. https://docs.aws.amazon.com/wellarchitected/latest/framework/welcome.html

[14] Google Site Reliability Engineering. Site Reliability Engineering: How Google Runs Production Systems. https://sre.google/sre-book/table-of-contents/

[15] Google Engineering Practices. How to Write Code Review Comments. https://google.github.io/eng-practices/review/reviewer/comments.html

[16] Google Site Reliability Engineering Workbook. Postmortem Practices. https://sre.google/workbook/postmortem-culture/

[17] OWASP Foundation. OWASP Code Review Guide. https://owasp.org/www-project-code-review-guide/

[18] OWASP Cheat Sheet Series. Secure Code Review Cheat Sheet. https://cheatsheetseries.owasp.org/cheatsheets/Secure_Code_Review_Cheat_Sheet.html

[19] GitHub Docs. About pull requests. https://docs.github.com/pull-requests/collaborating-with-pull-requests/proposing-changes-to-your-work-with-pull-requests/about-pull-requests

[20] NIST AI Resource Center. AI RMF Core: Govern, Map, Measure, Manage. https://airc.nist.gov/airmf-resources/airmf/5-sec-core/

[21] NIST AI Resource Center. Govern Playbook: Human-AI roles and oversight. https://airc.nist.gov/airmf-resources/playbook/govern/

[22] O'Connell, William T. Engineering Trustworthy Intelligent Systems (ETIS): Software Engineering, Governance, and Operational Trust in the AI Era. ETIS Framework, 2026. https://etisframework.org

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 Review and Readiness: Turning Inspection, Challenge, and Evidence into Trustworthy Change. ETIS White Paper Series, WP-007, Version 1.0.