ETIS WHITE PAPER SERIES

WP-005

Engineering Education in the AI Era

Preparing Students to Engineer, Verify, Govern, and Operate Intelligent Systems

CORE THESIS

AI does not reduce the need for software engineering education. It changes what credible evidence of engineering competence must look like.

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-005 | Version 1.0

Executive Abstract

Generative and agentic AI are changing the conditions under which software engineering is learned and practiced. Students can now produce code, tests, documentation, diagrams, and design alternatives at a speed that was recently unavailable to professional teams. Coding agents can inspect repositories, modify multiple files, run tests, and prepare pull requests. At the same time, employers are raising expectations for architecture, systems thinking, security, communication, judgment, and operational responsibility. The result is not the end of software engineering education. It is a demand for a more authentic form of it. [1-10]

Traditional computing courses often used the submitted artifact as a proxy for competence. In an AI-rich environment, that proxy is increasingly weak. A polished program may reveal little about whether the student understood the requirement, selected a sound design, verified generated work, managed risk, collaborated professionally, or can maintain and operate the result. The central assessment question therefore shifts from 'What did the student produce?' to 'What engineering claims can the student defend, and what evidence supports those claims?'

This paper argues that software engineering education should move toward repository-centered, evidence-centered, team-based learning. Students should use AI throughout the lifecycle, but accepted AI output must remain attributable, reviewable, testable, and owned by humans. Courses should preserve durable foundations while teaching students to specify intent, engineer context, supervise agents, evaluate behavior, govern authority, and operate systems. Assessment should examine decision quality, traceability, review discipline, testing, release readiness, and learning across iterations—not merely code volume or a successful demonstration.

Engineering Trustworthy Intelligent Systems (ETIS) provides one implementation of this educational direction. Its course model treats the repository as the operational engineering environment, assignments as phase gates, reviews as evidence-based defenses, and AI as an engineering participant whose work must be disclosed and verified. The objective is to prepare graduates who can move faster with AI without surrendering understanding, accountability, or professional judgment.

KEY IDEA

The educational challenge is not to prevent students from using AI. It is to ensure that AI use produces deeper engineering capability rather than synthetic performance.

1. A Profession and a Curriculum at an Inflection Point

Software engineering education has always evolved with professional practice. Structured programming changed what introductory programming meant. Object-oriented design reshaped architecture and modeling. Open source, cloud platforms, DevOps, and continuous delivery moved version control, automation, operations, and collaboration closer to the center of the curriculum. AI is another major transition, but its educational impact is unusually broad because it affects both the subject being taught and the means by which students complete the work.

The current generation of AI tools can explain code, generate implementations, draft requirements, propose designs, create tests, review pull requests, and operate across repositories. The technology is progressing from conversational assistance toward delegated engineering work. This means that assignments built primarily to

measure manual production are losing validity. A student may submit more code than prior cohorts while exercising less independent reasoning. Conversely, a student who uses AI effectively may perform more sophisticated analysis, test more failure modes, and learn faster than would have been possible through unaided work.

The correct response is neither prohibition nor uncritical adoption. Prohibition teaches students to work in an artificial environment that no longer resembles the profession. Uncritical adoption allows students to outsource the cognitive work that education is intended to develop. The educational design problem is to create conditions in which AI accelerates inquiry and production while the learner remains responsible for intent, selection, verification, explanation, and consequence.

This need is consistent with the direction of the discipline. CS2023 emphasizes competency, professional practice, ethics, security, and the integration of artificial intelligence across computing education. SWEBOK V4 expands the recognized body of software engineering knowledge to include architecture, operations, security, Agile, and DevOps. ABET continues to require programs to demonstrate student outcomes involving problem analysis, solution design, communication, ethics, teamwork, and disciplined computing practice. [1-4] AI changes the environment in which those outcomes are achieved; it does not make them obsolete.

2. The End of Artifact-as-Proof

For decades, educators could reasonably infer some level of competence from a completed artifact. A student who produced a functioning program probably wrote much of it, debugged it, and learned something about the underlying concepts. That inference was never perfect—students could copy, divide work unevenly, or rely on external help—but the effort required to create the artifact provided a meaningful signal.

Generative AI weakens that signal. Code, prose, tests, diagrams, and presentations can be produced quickly, often with professional polish. Detection is unreliable and increasingly irrelevant because AI-assisted work is normal in industry. The educational task is not to prove that no assistance occurred. It is to determine whether the student can act as the accountable engineer of the resulting system.

This changes assessment design. A working feature is evidence of capability, but it is not sufficient evidence of understanding. A repository may contain extensive documentation, but documentation created after the fact may not prove that decisions were made deliberately. A student may explain a design fluently with AI-generated language while being unable to diagnose a failure or evaluate an alternative. Educators therefore need multiple connected forms of evidence: evolving requirements, issue history, commits, pull requests, review comments, tests, architecture decisions, AI-use records, release baselines, runtime observations, postmortems, and oral defense.

The same principle applies to team projects. A polished team demonstration can conceal uneven participation, unreviewed generated code, weak traceability, or a system that cannot be reproduced. Repository evidence makes contribution, reasoning, and change history visible. Peer review and role ownership add context. A phase-gate discussion tests whether the team can connect claims to evidence without relying on private explanation.

The central educational distinction is between an artifact and an engineering claim. 'The feature works' is a claim. A demonstration, automated test, pull request, and release record may support it. 'The system is secure' is a much broader claim requiring threat analysis, permission design, dependency review, negative testing, and residual-risk disclosure. Teaching students to calibrate claims to evidence is itself a professional capability.

ASSESSMENT SHIFT

In the AI era, a submitted artifact is no longer reliable proof of learning. Competence must be demonstrated through connected evidence, explanation,

and accountable action.

Educational Question	Traditional Proxy	Stronger AI-Era Evidence
Did the student understand the problem?	Final requirements document	Requirements history, assumptions, questions, stakeholder rationale, and oral defense.
Did the student engineer the solution?	Working code	Architecture decisions, issue-to-PR traceability, review evidence, tests, and explanation of tradeoffs.
Was AI used responsibly?	Honor-code statement	AI-use log, accepted and rejected output, human edits, verification, and accountable owner.
Did the team collaborate professionally?	Shared final submission	Role ownership, issues, reviews, meeting evidence, contribution history, and peer evaluation.
Is the release trustworthy?	Successful demo	Controlled baseline, CI results, known limitations, runtime evidence, release notes, and residual risks.

3. Foundations Matter More When AI Produces the First Draft

A common student interpretation of AI is that foundational knowledge has become less important. The opposite is true. When AI generates the first draft, the learner must judge whether that draft is correct, appropriate, secure, maintainable, and consistent with the larger system. Judgment without foundations is imitation.

Students still need algorithms, data structures, complexity, programming-language concepts, databases, concurrency, networks, operating systems, architecture, security, testing, and human-computer interaction. They also need the ability to read unfamiliar systems and reason about state, failure, dependencies, and tradeoffs. AI can explain these concepts and accelerate practice, but it cannot assume professional responsibility for a system whose behavior the student does not understand.

The learning sequence should therefore change from 'master everything before using AI' to 'use AI inside a deliberately scaffolded progression.' Early exercises may constrain AI use to critique, explanation, or test generation so that students first develop mental models. Later work can permit broader generation while increasing expectations for review, evidence, and operational quality. The objective is not manual purity. It is progressive transfer of responsibility to the learner.

This distinction is particularly important for entry-level careers. Employers are automating some routine tasks while increasing the value of analytical thinking, technology literacy, security, communication, collaboration, and lifelong learning. The World Economic Forum reports strong growth in demand for AI, big data, and cybersecurity skills alongside durable human capabilities such as creative thinking, resilience, leadership, and collaboration. Its work on entry-level employment also warns that AI is changing the apprenticeship tasks through which new professionals traditionally learned. [7-9] Universities must therefore create authentic opportunities to practice judgment that the labor market may no longer provide automatically.

4. AI Literacy Is Necessary but Not Sufficient

AI literacy commonly includes understanding what generative models can do, recognizing hallucination and bias, writing effective prompts, protecting privacy, and evaluating output. These capabilities are necessary for all students. Software engineers need a broader competence: they must know how to integrate AI into a controlled engineering system.

That competence includes specification, context engineering, model and tool selection, repository instructions, evaluation design, security boundaries, provenance, cost awareness, human oversight, observability, and rollback. It also includes knowing when not to use AI. A responsible engineer distinguishes a low-consequence drafting task from a high-consequence decision, and adjusts verification and authority accordingly.

UNESCO's competency frameworks for students and teachers emphasize a human-centered mindset, ethics, AI techniques and applications, system design, and professional learning. UNESCO's guidance on generative AI in education similarly calls for human capacity, policy, privacy, and age-appropriate use. The U.S. Department of Education emphasizes human-centered design, enhanced feedback loops, explainability, and the need to keep educators in control. [5, 6, 10-12] These are essential foundations, but computing programs should extend them into discipline-specific engineering practice.

For software engineering students, disclosure should be treated as provenance rather than confession. The useful questions are what tool was used, for what task, what output was accepted, what was rejected, how the work was changed, what verification occurred, and who is accountable. This makes AI use reviewable and allows students to learn from both successful and failed assistance.

ETIS DOCTRINE

Prompt skill is not engineering competence. Engineering competence begins when the learner can define the task, control the context, verify the result, and own the consequences.

5. The Repository Becomes the Learning Environment

A learning management system is effective for schedules, readings, announcements, grades, and submission. It is not a realistic environment for learning professional software engineering. Modern engineering work happens through repositories, issues, branches, pull requests, automated checks, reviews, releases, and operational records. Students should learn inside that environment rather than merely upload its final products to an LMS.

Repository-centered learning makes the work observable. Requirements can link to issues; issues to branches and pull requests; pull requests to reviews and tests; tests to releases; releases to runtime evidence and postmortems. AI instructions, policies, and usage records can live beside the system they govern. The README becomes the front door, not a forgotten class requirement. The repository becomes both the engineering workspace and the evidence of learning.

This model also creates better context for AI. Coding agents perform more effectively when repositories contain current requirements, architecture, conventions, test commands, ownership, and decision history. Student documentation is no longer paperwork written only for the instructor. It becomes executable context that improves human and AI collaboration. Poorly maintained context creates visible consequences, which makes the educational lesson authentic.

Repositories also support continuity across assignments. Instead of creating six unrelated submissions, students mature one system through successive engineering stages. Early requirements constrain later architecture; architecture shapes implementation; tests support release claims; runtime evidence informs improvement. Students experience software engineering as an accumulating system of decisions and evidence rather than a sequence of disconnected documents.

6. Team-Based Engineering Must Remain Central

AI can increase individual production, but professional software remains a team activity. Systems cross disciplinary boundaries, integrate many components, and require independent review. Architecture, security, operations, governance, product intent, and stakeholder communication cannot be reduced to a single developer-agent interaction.

A strong educational team model gives every student development responsibility while assigning primary and backup ownership for team leadership, planning, architecture, quality, AI governance, repository operations, and release readiness. Roles are not titles awarded for status; they are accountability mechanisms. Students learn that engineering leadership involves making work visible, coordinating decisions, escalating risk, and ensuring that evidence exists before claims are made.

AI changes team dynamics in two opposing ways. It can reduce coordination cost by summarizing discussions, drafting tasks, identifying dependencies, and generating review candidates. It can also conceal weak collaboration by allowing one person to produce most artifacts or by flooding the team with generated work that others cannot review. Courses should therefore emphasize small batches, visible ownership, independent review, and shared understanding.

Peer review remains essential. Students should review requirements, architecture, code, tests, release evidence, and AI use—not merely each other's final prose. Review quality should be evaluated. A comment that says 'looks good' is not equivalent to a review that identifies assumptions, tests a claim, challenges complexity, or documents why a change is safe enough to merge.

7. Assessment Should Reward Engineering Judgment

Assessment drives behavior. If grades reward feature count, students will maximize visible functionality. If grades reward document volume, they will create artifact theater. If grades reward code written without AI, they will either hide AI use or practice an obsolete workflow. AI-era assessment should reward defensible engineering judgment and the quality of the evidence that supports it.

A balanced assessment model includes product capability, process discipline, technical quality, evidence traceability, review quality, responsible AI use, operational realism, and individual understanding. These dimensions should not be independent checkboxes. The important question is whether they form a coherent engineering argument. A team that documents many risks but never changes its design has not demonstrated risk management. A team with extensive tests that are unrelated to requirements has not demonstrated verification.

Oral review becomes more important, but it should not become an adversarial memory test. A phase-gate conversation asks students to navigate their own repository, explain a decision, show supporting evidence, acknowledge limitations, and respond to a realistic challenge. The instructor can ask what changed after review, what the team rejected from AI, which risk remains, or how the system would fail. This tests ownership and understanding while respecting the use of professional tools.

Individual accountability can be established through role evidence, commit and review history, short reflections, peer evaluation, and targeted questions. Metrics should be interpreted cautiously; commit counts and lines changed are not measures of value. The goal is to identify contribution, judgment, and learning—not to turn the repository into a surveillance system.

8. A Two-Cycle Course Creates Authentic Learning

One of the strongest educational structures is to require students to build and release twice. The first cycle asks whether the team can establish a controlled capability. The second asks whether the team can improve the system based on evidence. This creates a learning loop that a single final project cannot provide.

In Cycle 1, students define a bounded vertical slice, establish the repository, create requirements and plans, design the architecture, implement through reviewed changes, test the primary workflow, and defend a release baseline. The objective is not breadth. It is to demonstrate that the team can move from intent to a controlled release.

Cycle 2 begins with a postmortem and evidence review. Teams examine defects, weak tests, architecture friction, unclear permissions, poor observability, release gaps, and review findings. They then select a small number of maturity improvements. The best second cycle may add no major feature. It may instead strengthen recovery, security, runtime visibility, regression testing, traceability, or operational guidance.

This structure teaches an essential professional lesson: engineering maturity is not feature accumulation. It is the ability to learn from evidence and improve the weakest credible claim about the system. It also prevents the common student pattern of abandoning process once the demo works.

COURSE DESIGN PRINCIPLE

Cycle 1 asks, “Can it work under control?” Cycle 2 asks, “Can the team improve what the evidence says is weak?”

Course Stage	Student Work	Evidence of Learning
Launch	Team roles, project brief, repository, AI policy, initial requirements.	Visible ownership, professional workflow, explicit assumptions, and governed AI use.
Plan	Scope, work breakdown, estimates, risks, dependencies, and commitments.	Reasoned plan connected to capacity, uncertainty, and repository work.
Architect	Components, interfaces, data/context, decisions, boundaries, and review.	Defensible tradeoffs, review findings, and controlled architecture baseline.
Construct	Issues, branches, commits, PRs, CI, tests, AI-assisted review.	Traceable change, independent review, verification, and explanation.
Release	Baseline, release notes, known limitations, demo, peer review.	Claims connected to reproducible release and honest residual risk.
Mature	Postmortem, evidence-selected improvements, observability, runbook, final release.	Learning from evidence, operational thinking, and stewardship.

9. The Instructor Becomes the Architect of an Engineering Environment

AI does not make the instructor less important. It changes the instructor's highest-value work. Less class time should be spent transmitting information that students can retrieve on demand. More should be spent framing problems, designing constraints, modeling judgment, reviewing evidence, exposing tradeoffs, and helping students learn from failure.

The instructor must design the environment before the semester begins: project boundaries, repository starter structure, templates, examples, AI expectations, phase gates, review rubrics, and the relationship among the LMS, GitHub, and professional reference material. Good scaffolding reduces administrative confusion while preserving intellectual ambiguity. Students should not have to guess where evidence belongs, but they should have to decide what evidence is sufficient.

Faculty also need a clear AI stance. Vague statements such as 'use AI responsibly' are not enough. Students need examples of acceptable assistance, disclosure expectations, prohibited data, verification requirements, and the consequences of submitting work they cannot explain. Policies should distinguish learning exercises where unaided work is important from engineering work where AI use is expected.

Feedback should increasingly target reasoning and system quality. Instead of correcting every sentence, the instructor can challenge unsupported claims, missing links, weak tests, architectural drift, unreviewed changes, or hidden residual risk. AI can help instructors generate questions, compare evidence, and identify inconsistencies, but grading and consequential judgments should remain accountable human decisions.

10. Institutions Need Governance, Access, and Faculty Capability

Course redesign cannot be left entirely to individual instructors. Institutions need coherent guidance on approved tools, privacy, intellectual property, accessibility, data handling, academic integrity, procurement, and support. UNESCO reported in 2025 that many higher-education institutions had established or were developing AI guidance, reflecting a rapid shift from isolated experimentation toward institutional policy. [13]

Access is an educational issue. If advanced AI tools materially improve learning and productivity, unequal access can become unequal opportunity. Programs should decide which capabilities are institutionally provided, which are optional, and how assignments remain fair when students use different models. The goal need not be identical tools, but expectations must account for capability differences and cost.

Faculty development is equally important. Instructors need opportunities to understand AI limitations, redesign assessment, create discipline-specific policies, evaluate generated work, and use tools without surrendering professional judgment. UNESCO's teacher framework emphasizes a human-centered mindset, ethics, pedagogy, and continuing professional learning. These are not one-time training topics; model capabilities and educational risks will continue to change. [6]

Institutional governance should preserve experimentation. Overly centralized policy can freeze practice around today's tools, while complete decentralization creates inconsistency and risk. A better model establishes principles and protected boundaries—privacy, human accountability, accessibility, transparency, security—while allowing faculty to test new pedagogies and share evidence about what works.

11. A Graduate Capability Model for AI-Era Software Engineering

The goal of an AI-era software engineering program should not be to produce students who are expert users of a particular model. Tools will change too quickly. Programs should develop durable capabilities that remain valuable as assistance becomes more powerful and autonomy increases.

The graduate should be able to define intent and constraints; analyze stakeholders and risk; design architecture and authority boundaries; use repositories and collaborative workflows; employ AI across the lifecycle; verify code and behavior; communicate decisions; manage evidence; release controlled changes; observe runtime behavior; respond to failure; and explain the ethical and operational consequences of the system.

This model aligns traditional software engineering knowledge with emerging AI competence. SWEBOK, CS2023, SE2014, ABET outcomes, NIST's secure-development guidance, the ACM Code of Ethics, and UNESCO's AI competency work all reinforce different parts of this picture. [1-6, 14-17] ETIS adds an implementation model that connects them through lifecycle stages and repository evidence.

The most important capability is professional ownership. Graduates should be able to say: I know what this system is intended to do; I understand the important design decisions; I can show how changes were reviewed and tested; I know where AI contributed; I can identify limitations and residual risks; and I know what evidence would cause me to change my judgment.

Capability Domain	Graduate Can...	Representative Evidence
Intent and requirements	Translate ambiguity into outcomes, constraints, acceptance criteria, and authority boundaries.	Requirements, assumptions, open questions, impact and risk analysis.
Architecture and context	Design components, interfaces, data flows, context sources, controls, and failure behavior.	Architecture overview, ADRs, diagrams, context and permission model.
AI-assisted delivery	Use AI productively while preserving understanding, provenance, review, and ownership.	AI-use log, prompts/instructions where appropriate, edits, tests, rejected output.
Verification and assurance	Test correctness, security, policy adherence, failure paths, and nondeterministic behavior.	Automated tests, evaluation sets, CI results, review records, security evidence.
Operations and stewardship	Release, observe, recover, learn, and maintain trustworthy behavior over time.	Release baseline, runbook, observability, incident/postmortem, residual risks.
Professional collaboration	Lead, review, communicate, and make accountable decisions in a team.	Issues, PR reviews, role ownership, meeting decisions, peer feedback, oral defense.

12. The ETIS Educational Operating Model

Engineering Trustworthy Intelligent Systems (ETIS) treats education as professional formation rather than artifact completion. Its educational model is built on five connected ideas: repository-centered engineering, evidence-centered engineering, progressive lifecycle stages, governed AI assistance, and operational trust.

Repository-centered engineering gives students a realistic place to work. Evidence-centered engineering changes the basis of assessment from polish to defensibility. Lifecycle stages provide a progression without assuming a single development methodology. Governed AI assistance allows students to use modern tools while preserving provenance and accountability. Operational trust extends the definition of done beyond the demonstration to release, observation, recovery, and stewardship.

The course environment is intentionally layered. The LMS defines official course direction, timing, and submission. The team repository contains the authoritative engineering work and evidence. ETIS provides broader professional guidance and reusable reference patterns. AI tools assist throughout the lifecycle, but the team remains accountable for every accepted artifact and system behavior.

This model does not require every course to become an enterprise simulation. Scope should remain bounded. A small vertical slice engineered through two cycles can teach more than a large application built without discipline. The educational ambition lies in the quality of the engineering reasoning and evidence, not the size of the product.

ETIS EDUCATIONAL POSITION

The course project is not primarily a software product. It is a controlled environment in which students learn to make and defend professional engineering decisions.

13. A Maturity Path for Programs and Courses

Programs will adopt AI-era education at different speeds. A useful maturity model is progressive rather than binary. The objective is not to reach the highest level in every course, but to ensure that students encounter increasingly authentic responsibility across the curriculum.

At the first level, AI is unmanaged: policies are unclear, assessment assumes unaided work, and use is hidden. At the second, AI is permitted with basic disclosure, but courses remain artifact-centered. At the third, AI is integrated into selected learning activities and students are expected to verify output. At the fourth, courses use repositories, team workflows, phase gates, and connected evidence. At the fifth, programs coordinate competencies across courses and include operational, governance, and agentic-system responsibilities.

Maturity should not be measured by the number of AI tools deployed. It should be measured by whether students demonstrate stronger understanding, better feedback, more authentic practice, equitable access, reliable assessment, and professional accountability. Technology adoption that weakens those outcomes is not educational progress.

Level	Educational Pattern	Primary Risk / Opportunity
1. Reactive	AI use is discouraged or handled case by case.	Hidden use, obsolete assessment, inconsistent enforcement.
2. Disclosed	AI is allowed with general disclosure.	Transparency improves, but evidence of understanding remains weak.
3. Integrated	AI supports defined learning tasks with verification expectations.	Feedback and productivity improve; course design begins to adapt.
4. Evidence-Centered	Repositories, team workflow, reviews, phase gates, and AI provenance are assessed.	Authentic professional learning and defensible competence.
5. Programmatic	Capabilities progress across the curriculum, including agentic, governance, and operational responsibility.	Graduates prepared for AI-native engineering and continuing change.

14. Implications for the Next Decade

The near future will bring more capable coding agents, persistent project memory, autonomous test and review workflows, and tighter integration among repositories, issue trackers, cloud platforms, and enterprise systems. Students will increasingly work with AI agents that can perform multi-step tasks rather than merely suggest text. The educational response must scale with the authority of those systems.

Introductory courses will still teach programming, but they will need clearer decisions about when generation helps learning and when it bypasses it. Advanced courses will focus more on existing systems, architecture, integration, verification, security, context, and operations. Capstone experiences will resemble governed engineering environments, with AI agents participating under explicit rules and evidence requirements.

Assessment will become more continuous and portfolio-based. Institutions will rely less on isolated take-home artifacts and more on repository history, staged reviews, oral defense, controlled exercises, and evidence accumulated over time. This does not mean constant surveillance. It means aligning assessment with how engineering responsibility is actually demonstrated.

The strongest programs will also preserve human formation. Engineers need curiosity, humility, communication, empathy, resilience, and the courage to challenge unsafe decisions. The ACM Code of Ethics places the public good, avoidance of harm, honesty, quality, privacy, and professional competence at the center of computing practice. [16] AI makes those obligations more consequential, not less.

Conclusion: Teach Students to Own the System

Software engineering education should not be organized around a contest between students and AI. AI is becoming part of the professional environment, and students must learn to use it. The more important question is whether education develops engineers who can remain accountable when powerful systems generate, recommend, and act.

The durable foundations of the discipline remain essential. Requirements, architecture, programming, testing, security, teamwork, configuration management, deployment, operations, ethics, and maintenance do not disappear when AI accelerates production. They become the means by which AI-generated work is turned into trustworthy system behavior.

The educational model must therefore move beyond artifact submission. Students should work in repositories, build through reviewed changes, disclose meaningful AI use, connect claims to evidence, defend decisions, release controlled baselines, observe behavior, and improve systems through postmortem learning. They should experience the difference between a demo and operational proof before they enter industry.

The future engineer will not be distinguished by the ability to type code faster than a model. The future engineer will be distinguished by the ability to define what should be built, supervise how it is built, verify what was produced, govern what may act, and accept responsibility for the system's consequences. That is the standard software engineering education must now prepare students to meet.

References

[1] ACM, IEEE Computer Society, and AAAI. “Computer Science Curricula 2023 (CS2023).” 2024. Available online

[2] IEEE Computer Society. “Guide to the Software Engineering Body of Knowledge (SWEBOK Guide V4.0a).” 2025. Available online

[3] ABET. “Criteria for Accrediting Computing Programs, 2025-2026.” Available online

[4] ACM and IEEE Computer Society. “Software Engineering 2014: Curriculum Guidelines for Undergraduate Degree Programs in Software Engineering.” 2015. Available online

[5] UNESCO. “AI Competency Framework for Students.” 2024/2026. Available online

[6] UNESCO. “AI Competency Framework for Teachers.” 2024/2026. Available online

[7] World Economic Forum. “The Future of Jobs Report 2025.” Available online

[8] World Economic Forum. “Future of Jobs Report 2025: 78 Million New Job Opportunities by 2030, but Urgent Upskilling Needed.” January 7, 2025. Available online

[9] World Economic Forum. “Artificial Intelligence and the Future of Entry-Level Work.” 2026. Available online

[10] UNESCO. “Guidance for Generative AI in Education and Research.” 2023, updated 2026. Available online

[11] U.S. Department of Education, Office of Educational Technology. “Artificial Intelligence and the Future of Teaching and Learning.” 2023. Available online

[12] Stanford Institute for Human-Centered Artificial Intelligence. “The 2025 AI Index Report.” 2025. Available online

[13] UNESCO. “Two-Thirds of Higher Education Institutions Have or Are Developing Guidance on AI Use.” September 2, 2025. Available online

[14] NIST. “Secure Software Development Framework (SSDF) Version 1.1, SP 800-218.” 2022. Available online

[15] NIST. “Secure Software Development Practices for Generative AI and Dual-Use Foundation Models.” 2024. Available online

[16] ACM. “ACM Code of Ethics and Professional Conduct.” 2018. Available online

[17] ACM and IEEE Computer Society. “Software Engineering Code of Ethics and Professional Practice.” Available online

[18] DORA. “State of AI-Assisted Software Development 2025.” 2025. Available online

[19] DORA. “Balancing AI Tensions: Moving from AI Adoption to Effective AI Use.” March 10, 2026. Available online

[20] Stanford HAI. “Artificial Intelligence Index Report 2026: Economy.” 2026. Available online

[21] OpenAI. “Introducing Codex.” 2025. Available online

[22] GitHub. “About GitHub Copilot Coding Agent.” 2026. Available online

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

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

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

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

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

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

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

[30] O'Connell, William T. “ETIS WP-004: Engineering Agentic Software Systems.” 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 Education in the AI Era: Preparing Students to Engineer, Verify, Govern, and Operate Intelligent Systems. ETIS White Paper Series, WP-005, Version 1.0.