COMP 330 PROFESSIONAL PAPER SERIES

Using AI Professionally

From Fast Output to Responsible Engineering Judgment

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)

COMP-WP-004 | Version 1.0 | Fall 2026

Core Thesis

Professional AI use is not measured by how much work a tool produces. It is measured by whether the engineer can define the task, control the context, evaluate the result, preserve provenance, manage authority, and remain accountable for the outcome.

Purpose for COMP 330. This paper establishes how AI is expected to be used throughout the course: actively, transparently, critically, and under human engineering ownership.

Executive Abstract

Artificial intelligence has moved from optional assistance to a normal part of software engineering. Developers use AI to explore unfamiliar code, draft requirements, generate implementations, create tests, review changes, investigate defects, write documentation, and increasingly delegate bounded tasks to coding agents. The direction is not temporary. GitHub and OpenAI now describe systems that can inspect repositories, modify code, run tools and tests, and return pull requests for human review [4][5].

The professional question is therefore no longer whether engineers should use AI. It is whether they can use it without surrendering understanding, quality, security, or accountability. DORA's 2025 research characterizes AI as an amplifier of the surrounding engineering system and identifies a verification tax: time saved in generation may be re-spent auditing work that developers do not fully trust [1][2]. GitHub's own responsible-use guidance states that AI review should supplement rather than replace human review and that generated feedback must be verified [6].

This paper presents a professional operating model for AI-assisted engineering. It treats AI use as a lifecycle decision rather than a prompting technique. The model begins with task classification and risk, continues through intent, context, bounded authority, small-batch execution, independent verification, disclosure, review, and release, and ends with operational observation and learning. It distinguishes assistance from delegation, fluent output from evidence, and disclosure from confession.

For COMP 330 students, the standard is direct: AI tools are encouraged, but their work must remain attributable, reviewable, testable, and owned by the team. Students should use AI to increase the scope and quality of engineering—not to conceal the absence of understanding. The strongest AI user is not the person who accepts the most generated work. It is the person who can determine where AI adds value, where it should not be trusted, what evidence is needed, and how the result fits the complete system.

The Professional Test

Can you explain what the AI was asked to do, what context and authority it received, what it produced, what you accepted or rejected, how you verified it, and why you remain willing to own the result?

1. AI Is Now Part of the Engineering Environment

AI is no longer confined to autocomplete. Contemporary systems can analyze repositories, propose plans, change multiple files, run tests, review pull requests, and work asynchronously. OpenAI describes Codex as a software-engineering agent that can write features, fix bugs, answer questions about a codebase, and propose pull requests for review [5]. GitHub's coding-agent and review products similarly place AI inside the repository workflow rather than beside it [4][6].

This changes the engineering environment in two ways. First, the cost of creating a candidate solution falls. Second, the number of decisions hidden inside that candidate can increase. An agent may choose an algorithm, dependency, file boundary, command, test strategy, or error-handling approach without asking about each one. The output may be coherent even when the decisions are weak.

The consequence is that engineers must become better at framing work. A vague request such as "add authentication" is no longer merely incomplete; it may trigger rapid implementation of the wrong identity model, session policy, permission boundary, or data flow. Professional AI use begins before the tool is invoked—with explicit intent, constraints, acceptance conditions, and risk.

The technology will continue to evolve. The durable skill is not familiarity with one interface. It is the ability to govern increasingly capable tools as part of an engineering system.

Industry Direction

The interface is moving from "help me write this" toward "complete this bounded engineering assignment." That movement transfers authority, and every transfer of authority creates an obligation for stronger context, evidence, and control.

2. Four Modes of AI Participation

AI use should not be treated as one undifferentiated activity. Asking a model to explain a compiler error is materially different from allowing an agent to change a repository or production environment. The control burden should rise with the authority and consequence of the work.

COMP 330 uses four practical modes: assist, coordinate, execute, and operate. The progression is not a maturity score and it does not imply that greater autonomy is always better. It is an authority gradient.

Mode	What AI does	Human responsibility
Assist	Explains, drafts, suggests, summarizes, or generates within a human-controlled task.	Review every meaningful output before use; verify against authoritative sources and the system.
Coordinate	Proposes plans, decomposes work, identifies dependencies, or organizes lifecycle activity.	Approve intent, scope, sequencing, owners, acceptance criteria, and escalation points.
Execute	Performs bounded multi-step work in repositories, tools, or test environments.	Control identity, permissions, context, sandboxing, tests, provenance, review, and merge authority.
Operate	Acts in or on behalf of a live system or business process.	Require runtime policy, observability, intervention, rollback, and accountable operational ownership.

Most student work will occur in the first three modes. Operation may be simulated or tightly bounded. What matters is that the team recognizes when a tool has moved from producing advice to exercising authority. A generated paragraph can be rejected easily. A schema migration, dependency update, or infrastructure change can create consequences that persist after the prompt is forgotten.

The correct posture is neither automatic trust nor automatic prohibition. It is progressive delegation: grant the minimum authority needed for the task, observe the result, verify the evidence, and expand authority only when the workflow proves capable of supporting it.

3. Begin with the Engineering Task, Not the Prompt

Prompt quality matters, but the prompt is not the engineering problem. Professional use begins by defining the task. What outcome is needed? Which requirement or issue authorizes it? Which constraints apply? What is outside scope? What evidence would demonstrate completion? What could go wrong?

This framing improves both human and AI performance. The tool receives a bounded assignment rather than an ambiguous wish. The engineer receives a basis for review rather than a polished artifact with no acceptance standard.

A strong task specification identifies the current system state, relevant architecture, authoritative sources, required interfaces, prohibited changes, test expectations, operational concerns, and the point at which the AI must stop and ask for guidance. For low-risk work, this may be a short issue description. For consequential work, it may require requirements, an ADR, threat considerations, data boundaries, and an explicit review plan.

The task should also make uncertainty visible. An AI system may fill missing information with plausible assumptions. Engineers should instead identify the unresolved questions and decide whether the agent may proceed with a documented assumption or must escalate.

Task Framing Principle

A weak prompt asks for an artifact. A professional engineering assignment defines the outcome, constraints, authority, evidence, and stopping conditions.

4. Context Is a Controlled Engineering Asset

AI output is shaped by the context it receives. In software engineering, that context may include requirements, repository structure, architecture decisions, source code, test commands, coding standards, policies, examples, tool descriptions, issue history, and operational evidence. More context is not automatically better. The objective is authoritative, relevant, current, and sufficiently bounded context.

Poor context creates predictable failures. Stale documentation produces outdated changes. Missing architecture causes local solutions that weaken the system. An untrusted web page or retrieved document may manipulate tool use. A tool description may imply authority the agent should not possess. A large context package may bury the instruction that matters most.

DORA's current capability guidance describes AI-accessible internal data as a means of connecting AI systems to proprietary codebases, documentation, and operational information through context engineering [3]. The principle is important for students: the quality of your repository affects the quality of both human and AI work. Current requirements, readable code, clear README guidance, useful tests, and explicit decisions are productivity infrastructure.

Context must also respect privacy and security. Do not place secrets, protected data, private credentials, or restricted information into tools without authorization. The fact that a tool can accept data does not mean you are permitted to provide it.

Context question	Professional expectation
Authority	Use sources recognized as authoritative for the decision.
Freshness	Confirm that instructions, dependencies, policies, and examples are current.
Scope	Provide the minimum sufficient context; exclude unrelated or sensitive information.
Conflict	Identify contradictory requirements or sources rather than allowing the model to choose silently.
Provenance	Preserve where material facts, instructions, and generated work came from.
Security	Treat retrieved content and tool output as untrusted until validated.

5. Generation Is Not Verification

AI can produce code, tests, explanations, and review comments. It cannot turn its own output into proof merely by producing more output. A generated implementation and generated test may share the same misunderstanding. A generated explanation may restate the code rather than validate the requirement. An AI reviewer may miss the same context-specific risk as the AI author.

DORA reports a verification tax in AI-assisted development: time saved during creation may be re-spent checking the result, and a significant share of developers report limited trust in generated code [2]. This is not a reason to avoid AI. It is a reason to design verification into the workflow rather than treating it as cleanup.

Verification should be independent enough to challenge the failure mode. Requirements should be checked against stakeholder intent. Code should be tested with focused unit and integration tests, static analysis, security checks, and review by someone who understands the surrounding system. Architecture should be challenged through alternatives and failure scenarios. Documentation should be compared with the implementation. High-risk behavior may require adversarial cases, operational simulation, or human subject-matter review.

GitHub's responsible-use guidance is explicit: Copilot code review should supplement human review, not replace it, and its feedback must itself be reviewed and verified [6]. The same rule applies to every AI-generated artifact.

Verification Rule

Never use the same unchallenged reasoning path as both producer and proof. Independent evidence requires a different test, source, tool, reviewer, or failure mode.

AI contribution	Minimum professional verification
Explanation or summary	Compare with authoritative documentation, source code, or primary evidence.
Requirement or user story	Validate stakeholder intent, scope, constraints, and acceptance criteria.
Code change	Inspect the diff; run focused tests; check architecture, security, maintainability, and failure behavior.
Generated tests	Confirm the oracle; add negative and edge cases not derived only from the implementation.
Architecture proposal	Compare alternatives; test assumptions, boundaries, dependencies, and operational consequences.
Review comments	Treat as leads for investigation; verify each material claim before acting.
Agent-executed task	Review the plan, commands, changed assets, tool calls, tests, provenance, and resulting state.

6. Work in Small Batches

AI makes large changes easy to create. That does not make them easy to understand. Large batches combine many assumptions, increase reviewer fatigue, hide causal relationships, and make rollback harder. DORA links AI-generated code with the danger of larger batch sizes and emphasizes small batches as a countermeasure for reviewability and stability [1][7].

A professional AI workflow keeps the unit of change bounded. Ask for one coherent outcome. Inspect the plan before implementation. Review the diff before requesting additional changes. Run tests at each step. Commit or checkpoint known-good states. Use branches and pull requests so generated work cannot silently become part of the controlled baseline.

Small batches also improve learning. When a result is wrong, the team can identify which assumption or instruction caused the failure. When many generated changes arrive together, the team may know that something broke without knowing why.

Faster generation should shorten the feedback loop, not enlarge the change set. The goal is more verified iterations, not more unreviewed output.

7. Disclosure Is Provenance, Not Confession

AI use should be visible because it affects how work is reviewed. Disclosure is not an admission of wrongdoing. It is provenance: information about how an artifact or decision was produced.

A useful AI-use record identifies the tool or agent, the task, the context or sources supplied, the level of authority, the material output, what humans accepted, changed, or rejected, the verification performed, and the accountable owner. The record should be proportionate. A minor wording improvement does not require the same detail as an agent-generated authentication change.

Provenance protects the student and the team. It makes contribution visible, allows reviewers to focus on likely failure modes, and creates a record of judgment rather than passive acceptance. It also supports learning. Rejected output and failed approaches may reveal more about engineering maturity than a polished final artifact.

Concealing AI use weakens the engineering record and makes ownership harder to establish. Overdocumenting every trivial interaction creates noise. The objective is material transparency: preserve the information another competent engineer would need to understand and review the work.

Record element	Question answered
Tool and mode	What system was used, and was it assisting, coordinating, or executing?
Task	What engineering outcome was the AI asked to advance?
Context	Which sources, files, instructions, or constraints shaped the output?
Disposition	What was accepted, modified, rejected, or deferred?
Verification	How was the material contribution challenged?
Ownership	Which student or team role remains accountable?

8. Security, Privacy, and Intellectual Property

Professional AI use includes obligations beyond correctness. Source code, credentials, customer data, personal information, security findings, and proprietary documents may be restricted even when a tool is technically available. Students must follow course policy, university policy, repository permissions, licenses, and the terms governing the AI service being used.

Do not place secrets in prompts, code, screenshots, logs, or repositories. Do not upload data you are not authorized to disclose. Do not assume that generated code is free of licensing, attribution, or dependency concerns. Review imports, packages, copied patterns, and suggested libraries. Scan dependencies and preserve license information where relevant.

AI-generated code can also introduce insecure defaults: missing validation, excessive permission, unsafe deserialization, weak error handling, exposed logs, or fabricated security APIs. Security review should be part of ordinary engineering, not a final pass. Least privilege, input validation, secure secret handling, and clear trust boundaries remain human responsibilities.

When uncertainty exists, stop and ask. Responsible engineering includes recognizing when the risk exceeds the team's authority or expertise.

Boundary Principle

Convenience is not authorization. A tool's ability to access, store, transform, or generate information does not establish your right to provide or use it.

9. AI Can Review, Critique, and Teach—But It Cannot Own the Decision

AI is valuable not only as a generator but also as a critic. It can propose edge cases, identify suspicious patterns, compare code with a requirement, explain unfamiliar APIs, generate counterarguments, or suggest simpler alternatives. Used this way, AI can broaden the questions a team asks.

The team should deliberately separate generation from critique. After producing a solution, change the instruction: ask for failure modes, security concerns, unnecessary complexity, missing tests, and reasons the approach may be wrong. Where possible, use a different source, tool, reviewer, or method. Skeptical review is more valuable than asking the system to praise its own work.

AI can also function as a tutor. Ask it to explain a concept at multiple levels, compare approaches, create practice scenarios, or question your understanding. But use the explanation to deepen your mental model, not to replace it. Verify unfamiliar claims with primary documentation and test them in code.

Ultimately, the merge, release, and submission decisions remain human decisions. AI can contribute evidence and recommendations. It cannot accept the professional consequence.

10. Failure Patterns That Look Productive

AI failure in student work often looks like success. The repository grows quickly. Documentation appears polished. Tests pass. The presentation sounds professional. Yet no one can explain the design, the tests only confirm the implementation, the architecture was never discussed, and the team discovers late that the system solves the wrong problem.

The most common failure is passive acceptance. A student asks for a complete solution, copies it into the project, fixes visible errors, and assumes the work is finished. Another is AI ping-pong: repeatedly pasting errors back into the tool without developing a model of the system. A third is generated theater: producing large quantities of requirements, risks, diagrams, and tests that did not influence decisions.

Team-scale failure occurs when AI increases output beyond review capacity. One member generates large changes while others become nominal reviewers. The team appears fast until integration, presentation, or failure exposes the lack of shared understanding.

Another failure is overtrust in AI review. A green pipeline and favorable AI comments can coexist with a wrong requirement, insecure permission model, poor user experience, or unrecoverable deployment. Automated signals support judgment; they do not replace it.

Weak pattern	Professional correction
Copy, compile, submit	Restate the requirement, inspect the design, verify behavior, and explain every material decision.
Prompt until errors disappear	Diagnose root cause with logs, tests, documentation, and a system model.
Generate the entire feature at once	Decompose into small, reviewable changes with explicit acceptance conditions.
Let AI write and review the same work	Add independent tests, human review, and different failure-oriented analysis.

Hide AI use	Preserve material provenance and make human judgment visible.
Produce documents after the fact	Use artifacts during the work so they influence decisions and become credible evidence.

11. The COMP 330 AI-Assisted Engineering Workflow

COMP 330 expects AI to be integrated into normal team workflow rather than used privately and revealed only at submission. The exact tools may vary, but the operating sequence should remain disciplined.

The workflow begins with an authorized work item. The team clarifies intent and acceptance criteria, identifies constraints and risk, and decides which AI mode is appropriate. Context is assembled from current repository sources. Work proceeds in a bounded branch or environment. Generated output is reviewed incrementally. Verification combines automated checks with human analysis. Material AI use is recorded. The pull request presents the engineering argument. A human team member accepts responsibility for merge and release.

This workflow is intentionally compatible with professional practice. It allows the team to move faster while preserving traceability, review, and evidence. It also creates a useful record for phase-gate presentations and individual learning.

Step	Expected evidence
1. Define the work	Issue, requirement, scope, acceptance criteria, owner, risk.
2. Select the AI role	Assist, coordinate, execute, or operate; authority and limits stated.
3. Supply context	Authoritative files, instructions, standards, and prohibited changes.
4. Execute in a small batch	Branch, task trace, generated or changed assets, commands and tool use.
5. Verify independently	Tests, analysis, security checks, comparison, reviewer challenge.
6. Record provenance	Tool, task, context, accepted/rejected output, human changes, owner.
7. Review and integrate	Pull request, comments, finding disposition, accountable merge decision.
8. Observe and learn	Defects, runtime evidence, postmortem findings, updated context or controls.

Course Doctrine

Use AI aggressively enough to increase what your team can accomplish. Govern it rigorously enough that your team can still explain, defend, and own everything it accepts.

12. What Professional AI Capability Looks Like

Professional capability is visible in decisions. You choose AI where it improves the work and decline it where the risk, privacy, uncertainty, or learning objective makes another approach better. You provide focused context rather than dumping the entire problem into a chat. You compare alternatives. You detect when output is plausible but unsuitable. You create evidence that is independent of the generated claim.

You also preserve understanding. You can explain the code, architecture, tests, limitations, and operational behavior without reopening the conversation that produced them. Your teammates can review the work. Another engineer can reconstruct the change from the repository. The system remains maintainable after the tool session ends.

This capability will outlast any product. Models, interfaces, and vendors will change rapidly. The durable professional skill is governing intelligent assistance inside a disciplined engineering process.

Career Signal

AI familiarity is becoming common. Responsible delegation, verification, context design, and accountable integration are the differentiators.

Before You Use AI on a COMP 330 Task

Pause long enough to answer these questions. The goal is not paperwork; it is to prevent fast output from outrunning engineering judgment.

- ✓ What requirement, issue, or team decision authorizes this work?
- ✓ What is the smallest useful outcome?
- ✓ Which AI mode is appropriate, and what authority will the tool receive?
- ✓ Which repository sources and constraints are authoritative?
- ✓ What information must not be disclosed?
- ✓ How will the team independently verify the result?
- ✓ What material AI use must be recorded?
- ✓ Who will review and own the accepted work?
- ✓ How can the change be stopped, reversed, or corrected if it is wrong?

Conclusion

AI is becoming a normal participant in software engineering. It can expand a team's capacity, shorten feedback, and make ambitious work feasible. It can also create more change than a team understands, more artifacts than it can validate, and more confidence than the evidence supports.

The difference is professional discipline. Engineers define the task, control context, bound authority, work in small batches, verify independently, preserve provenance, review honestly, and remain accountable for the result. They use AI as leverage without allowing it to become invisible authority.

That is the standard for COMP 330. The course does not ask you to prove that you can work without AI. It asks you to prove that you can engineer responsibly with it.

References and Further Reading

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

[2] DORA. Balancing AI Tensions: Moving from AI Adoption to Effective Use, 2026. https://dora.dev/insights/balancing-ai-tensions/

[3] DORA. AI-accessible Internal Data, 2026. https://dora.dev/capabilities/ai-accessible-internal-data/

[4] GitHub. GitHub Copilot Coding Agent and Responsible Use Guidance. https://docs.github.com/en/copilot/responsible-use/agents

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

[6] GitHub. Application Card: GitHub Copilot Agents—Responsible Use. https://docs.github.com/en/copilot/responsible-use/agents

[7] DORA. Working in Small Batches. https://dora.dev/capabilities/working-in-small-batches/

[8] GitHub. Review AI-generated Code. https://docs.github.com/en/copilot/tutorials/review-ai-generated-code

[9] National Institute of Standards and Technology. Artificial Intelligence Risk Management Framework (AI RMF 1.0), 2023. https://nvlpubs.nist.gov/nistpubs/ai/nist.ai.100-1.pdf

[10] National Institute of Standards and Technology. Artificial Intelligence Risk Management Framework: Generative Artificial Intelligence Profile, NIST AI 600-1, 2024. https://nvlpubs.nist.gov/nistpubs/ai/NIST.AI.600-1.pdf

[11] NIST AI Resource Center. AI RMF Playbook. https://airc.nist.gov/airmf-resources/playbook/

[12] William T. O'Connell. Engineering Trustworthy Software in the AI Era. ETIS White Paper WP-001, 2026. https://etisframework.org

[13] William T. O'Connell. Engineering Evidence. ETIS White Paper WP-003, 2026. https://etisframework.org

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

[15] William T. O'Connell. Context Engineering. ETIS White Paper WP-009, 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 Engineering Trustworthy Intelligent Systems (ETIS). He brings four decades of experience in software engineering, enterprise architecture, product and platform development, consulting delivery, quality, and technology leadership.

His career includes work across AT&T Bell Laboratories, IBM Research, IBM product development, enterprise software, major client modernization programs, and global technical leadership. He served as an IBM Distinguished Engineer and as CTO for IBM Consulting Quality and Delivery. His teaching focuses on helping students combine strong technical foundations with professional judgment, responsible AI use, evidence, teamwork, and operational accountability.