COMP 330 PROFESSIONAL PAPER SERIES

Working Effectively on an Engineering Team

Shared Ownership, Professional Challenge, and Human-AI Collaboration

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-003 | Version 1.0 | Fall 2026

Core Thesis

A strong engineering team is not a group of individuals dividing tasks. It is a system of shared intent, explicit ownership, visible work, disciplined review, credible evidence, and mutual accountability. AI can expand team capacity, but only teams that can coordinate, challenge, and understand the resulting work will convert that capacity into trustworthy outcomes.

Purpose for COMP 330. This paper establishes the professional team model you will use throughout the semester: every member contributes technically, roles create accountability, reviews create learning, the repository preserves shared understanding, and the team—not an individual or an AI tool—owns the result.

Executive Abstract

Software engineering is fundamentally collaborative. Modern systems cross technical domains, organizational boundaries, and operational environments. No single engineer can hold every requirement, design decision, implementation detail, risk, and production consequence in private memory. Professional teams succeed by turning individual knowledge into shared understanding and individual effort into controlled, reviewable change.

Artificial intelligence makes this team capability more important. AI assistants and coding agents can produce plans, code, tests, reviews, and documentation at a pace that can exceed the human team's ability to absorb them. Used well, AI reduces routine effort, expands exploration, and increases the team's reach. Used poorly, it creates hidden work, review overload, shallow agreement, duplicated effort, and artifacts that no one truly owns. DORA's research emphasizes that AI amplifies the surrounding sociotechnical system and that small batches, fast feedback, and strong underlying practices remain critical [1][2].

This paper presents a professional model for engineering teamwork. It explains why teams require shared intent, explicit decision rights, active roles, small-batch workflow, meaningful pull-request review, psychological safety, professional communication, conflict discipline, evidence-based decisions, and durable repository context. It also addresses individual accountability inside team work, how to prevent uneven contribution, and how human-AI teams should preserve ownership and independent judgment.

For COMP 330, the practical standard is direct: every student is a developer, assigned roles create primary ownership rather than exclusive control, important work remains visible in GitHub, consequential changes receive review, and each team member must understand enough of the system to explain and defend it. The goal is not simply to complete a group project. It is to learn how professional teams create trustworthy change together.

The Team Question

Can another competent member of the team understand the current intent, identify who owns the next decision, review the proposed change, find the supporting evidence, and continue the work without reconstructing it from private messages?

1. A Team Is an Engineering System

A team is often described as a collection of people with complementary skills. That definition is incomplete for software engineering. An engineering team is also a system for converting uncertain intent into coordinated decisions, implementation, evidence, and operational responsibility. Its interfaces include meetings, issues, pull requests, architecture records, tests, dashboards, and escalation paths. Its state includes priorities, decisions, unresolved risks, current work, and shared knowledge.

Like any system, a team can fail at its interfaces. Product intent may not reach implementation. A decision may remain in one person's head. Two developers may solve the same problem differently. A reviewer may approve work without understanding it. Operations may inherit a release with no context. These are not merely interpersonal problems; they are engineering failures in coordination and information flow.

High-performing teamwork therefore requires deliberate design. The team needs a common purpose, bounded work, visible ownership, dependable handoffs, structured challenge, and feedback from results. Roles and workflow should reduce ambiguity rather than create hierarchy for its own sake. The repository should preserve the team's durable memory so that progress does not depend on who happens to be present.

The team is also sociotechnical: people, AI tools, automation, repositories, and platforms interact. A change in one element changes the behavior of the whole. Adding a coding agent, for example, may increase production capacity while creating a review bottleneck. The correct question is not whether the tool makes one person faster. It is

whether the complete team becomes more effective, more informed, and more capable of producing reliable outcomes.

2. Shared Intent Comes Before Task Division

Weak teams begin by dividing tasks. Strong teams begin by aligning on the outcome. Before deciding who will build the interface, API, database, or tests, the team must understand the problem, users, constraints, acceptance conditions, and boundaries of the release. Otherwise, task division simply distributes different interpretations of the same ambiguity.

Shared intent does not mean that every member holds identical detail. It means that the team agrees on the purpose, current scope, most important risks, and definition of success. Members should understand how their work connects to the end-to-end workflow and which assumptions could invalidate another person's work.

This is why requirements and planning are team activities rather than documents produced by one role owner. The project manager may organize the plan, and the requirements owner may maintain the artifacts, but architecture, testing, implementation, and release all depend on the same intent. The team should challenge unclear terms, surface conflicting assumptions, and record decisions before implementation makes them expensive.

AI can help teams summarize discussion, identify missing acceptance criteria, decompose work, and propose alternatives. It can also create a polished plan that hides disagreement. A generated plan becomes team intent only after people examine it, modify it, and explicitly accept its assumptions.

3. Roles Create Accountability, Not Silos

Professional teams use roles because important concerns need owners. A team lead coordinates direction and removes barriers. A project or planning owner makes work and dependencies visible. An architecture owner protects structural coherence. A quality or test owner ensures that claims receive verification. A repository or release owner protects workflow and baselines. An AI-governance owner helps preserve provenance, verification, and responsible use.

These roles do not transfer the entire concern to one person. The architecture owner is not the only person allowed to think about architecture. The test owner is not the only person who writes tests. The team lead is not the person who makes every decision. A role creates primary responsibility for ensuring that the concern is addressed, integrated, and evidenced; the team remains collectively responsible for the system.

Role silos are dangerous because they create handoff behavior: "I finished my part; the rest is someone else's problem." Software does not respect those boundaries. A database decision affects APIs, performance, tests, security, and deployment. An interface choice affects user behavior and error recovery. Every member should understand the adjacent concerns that shape their work.

Backup ownership matters as well. If only one person can explain or operate a component, the team has created a single point of failure. Pairing, review, shared documentation, and rotation of selected responsibilities improve continuity without eliminating clear accountability.

Role pattern	Professional meaning
Primary owner	Ensures the concern is planned, maintained, reviewed, and evidenced.
Contributors	Perform work and bring domain knowledge; responsibility is shared.
Independent reviewer	Challenges claims and supplies perspective not owned by the author.
Backup owner	Maintains continuity and reduces single-person dependency.
Escalation authority	Resolves decisions beyond the role owner's delegated authority.

4. Visible Work Is the Foundation of Coordination

Teams cannot coordinate work that remains invisible. Private to-do lists, direct messages, verbal agreements, and code developed for long periods without integration create uncertainty about current state. Others cannot discover dependencies, challenge assumptions, or help until the work is nearly complete.

In repository-centered engineering, issues describe bounded work, ownership, acceptance conditions, dependencies, and status. Branches isolate change. Pull requests expose proposed work before it becomes part of the shared baseline. Reviews and automated checks record what was challenged. Releases identify what the team accepted. This workflow turns activity into shared, inspectable progress.

Visible work is not surveillance. Metrics such as commit count or lines changed are weak measures of value and should not become substitutes for judgment. Visibility exists so the team can coordinate, assist, review, and learn. The most important signal is whether another member can understand what is happening and what decision is needed next.

Small batches are essential. DORA identifies working in small batches as a mechanism for faster feedback and as an important safety net when AI increases development velocity [1][2]. A small change is easier to understand, test, review, revise, and recover. Large hidden changes increase sunk cost and make weak review more likely.

5. Review Is How Teams Think Together

A pull-request review is not an administrative approval. It is the point at which individual work becomes shared engineering responsibility. GitHub describes pull requests as the foundation for discussing and reviewing changes before merge, and its review model supports comments, approvals, requested changes, and responsible code owners [3][4].

A useful review examines intent, design, correctness, tests, maintainability, security, operational effect, and evidence. The reviewer should understand why the change exists, not merely inspect the diff. The author should make the change reviewable through a clear description, bounded scope, linked issue, test evidence, and honest statement of limitations.

Review also transfers knowledge. The reviewer learns the component, the author clarifies reasoning, and the repository preserves the discussion. A team that treats review as delay loses this benefit. A team that reviews only after a large change is finished creates unnecessary conflict because the implementation has already accumulated cost.

AI can support review by finding patterns, proposing tests, identifying suspicious changes, or comparing implementation with requirements. It should not become the sole reviewer of AI-generated work. Shared models and context can create correlated blind spots. Human reviewers remain responsible for system fit, tradeoffs, and consequences.

Review Standard

A review is complete only when the reviewer understands the claim, examines the relevant evidence, records meaningful findings, and the team dispositions those findings.

6. Psychological Safety Enables Engineering Honesty

Engineering teams need members to say “I do not understand,” “I think this is unsafe,” “the estimate was wrong,” “the test is weak,” or “my implementation failed.” When people believe these statements will lead to embarrassment, exclusion, or retaliation, the team loses information precisely when it is most valuable.

Psychological safety is not the absence of standards or disagreement. It is the expectation that people can raise questions, admit uncertainty, and challenge work without being treated as disloyal or incompetent. Google's team-effectiveness work made psychological safety widely visible as a foundational condition for team performance and learning [5].

Safety must coexist with accountability. A safe team can still expect preparation, follow-through, technical quality, and professional conduct. The distinction is between challenging the work and diminishing the person. "This change lacks recovery evidence" is a professional finding. "You always write bad code" is personal and destructive.

Leaders and role owners shape this environment through their response to bad news. If the first person to expose a defect is blamed, defects will remain hidden. If uncertainty is welcomed and converted into action, the team becomes more reliable. The same principle applies to instructors and reviewers: rigorous challenge should increase learning rather than reward performance theater.

7. Conflict Is Necessary; Dysfunction Is Not

Disagreement is unavoidable in engineering because systems involve tradeoffs. Teams must choose between speed and flexibility, consistency and availability, reuse and simplicity, feature scope and quality, automation and human control. Avoiding disagreement does not remove the tradeoff; it allows an unexamined decision to win.

Professional conflict focuses on claims, constraints, evidence, and consequences. Members should state the decision being made, identify the alternatives, explain the relevant criteria, and distinguish fact from preference. Architecture decision records are useful because they preserve context, alternatives, rationale, and consequences after the conversation ends.

When the team cannot agree, the escalation path should be known. Some decisions belong to a role owner within defined authority. Others require the team lead, instructor, product owner, security owner, or release authority. Escalation is not defeat; it is a mechanism for resolving decisions whose consequence exceeds the local team's authority or available knowledge.

Dysfunctional conflict becomes personal, repetitive, hidden, or disconnected from evidence. It may appear as passive resistance, last-minute objections, private coalitions, or unilateral implementation. The remedy is not forced harmony. It is a transparent decision process with ownership and closure.

Constructive conflict	Dysfunctional conflict
Challenges the idea	Attacks the person
Uses explicit criteria	Uses status or persistence
Preserves alternatives and rationale	Reopens closed decisions without new evidence
Escalates transparently	Builds private coalitions
Ends with disposition and ownership	Ends with ambiguity or resentment

8. Communication Is an Engineering Deliverable

Professional communication is not ornamental. It determines whether requirements are understood, decisions can be reviewed, handoffs preserve context, incidents are managed, and stakeholders can act. CS2023 treats professional communication as a core concern because computing professionals communicate with audiences that have different goals and levels of technical knowledge [6][7].

Good engineering communication is concise enough to use and complete enough to prevent consequential ambiguity. An issue should explain the outcome and acceptance conditions. A pull request should explain why the change deserves integration. A meeting record should preserve decisions and actions rather than transcribe conversation. A release note should tell users and operators what changed and what remains limited.

Different audiences require different explanations. A developer may need interface details. An executive needs risk, outcome, and decision. An operator needs failure signals and recovery steps. A user needs understandable behavior and limitations. Speaking professionally means selecting the information necessary for the audience to make its decision.

AI can improve clarity, summarize discussion, and help tailor communication. The author remains accountable for accuracy. A polished summary that changes the meaning of a decision is worse than an imperfect note that preserves it.

9. Individual Accountability Inside Team Work

Team ownership does not eliminate individual accountability. Every member should make meaningful technical contributions, fulfill assigned role obligations, participate in review, communicate risks, and understand the work they accept. A team project should not become one person building while others produce peripheral documents.

Contribution cannot be measured responsibly through a single number. Commit counts, issue counts, or lines changed may provide context but do not reveal difficulty, quality, review, leadership, or invisible coordination. Individual evidence should include implemented work, reviews performed, decisions influenced, problems resolved, role outcomes, and the ability to explain the resulting system.

When a member is blocked or falling behind, early visibility allows the team to adjust scope, redistribute work, pair, or seek help. Silence converts a manageable problem into a late team failure. Professional accountability includes asking for help before the consequence becomes unavoidable.

Peer evaluation should describe observable behavior and impact rather than popularity. Useful evidence includes dependability, technical contribution, review quality, communication, ownership, willingness to help, and responsiveness to feedback. Honest peer feedback is part of the team's learning, not a mechanism for retaliation.

Individual Standard

You are accountable for your own contribution and for the work you approve. "The team did it" and "AI generated it" are not substitutes for professional ownership.

10. Human-AI Teamwork Requires Explicit Boundaries

AI is becoming part of the engineering team's operating environment. It may assist an individual, coordinate tasks, produce a multi-file change, review work, or execute bounded steps. The metaphor of a digital teammate is useful only when the authority and accountability remain clear.

An AI system does not own a requirement, accept a risk, approve a release, or bear consequences. The human team decides what task is delegated, what context is supplied, which tools may be used, what evidence is required, and who accepts the result. AI use should remain visible enough that reviewers can understand what was generated and how it was verified.

Human-AI collaboration can fail through overtrust, hidden prompts, copied output, duplicated agent work, excessive generation, and review overload. It can also create false independence when one agent generates work and another similar agent approves it. Strong teams use AI for leverage while preserving small batches, human challenge, independent tests, and explicit ownership.

The team should judge AI by system outcome, not by conversational quality. Did it reduce cycle time without increasing defects? Did it create useful tests? Did it surface a risk? Did it make the change easier or harder to review? Did the team learn? These are engineering questions, not product enthusiasm.

11. The COMP 330 Team Operating Model

COMP 330 gives you a controlled environment in which to practice professional team behavior. Your team will maintain one operational repository, mature one system across two cycles, use assigned roles, and accumulate evidence through issues, decisions, pull requests, tests, reviews, releases, and postmortems.

Every member is expected to contribute to implementation. Roles add primary ownership for coordination, architecture, quality, AI use, repository workflow, and release readiness. No role excuses a student from understanding the system or reviewing the work of others.

Team meetings should produce decisions, actions, owners, and evidence updates. Work should be broken into reviewable issues. Pull requests should remain small enough for meaningful review. Important changes should not be merged solely because a deadline is near. Risks and limitations should be visible before the presentation.

At each phase gate, the team should be able to navigate the repository and defend its claims. The strongest team is not necessarily the team with the most features. It is the team whose members share understanding, whose work is reviewable, whose evidence is credible, and whose next improvement is informed by what the current system revealed.

Team practice	Evidence in COMP 330
Shared intent	Current project brief, requirements, acceptance conditions, visible scope
Visible work	Issues, assignments, branches, pull requests, project status
Structured challenge	Review comments, requested changes, meeting and decision records
Quality ownership	Tests, defects, CI results, release criteria, known limitations
AI accountability	AI-use record, human changes, verification, reviewer acceptance
Learning	Postmortem, corrective work, Cycle 2 maturity improvements

12. A Professional Standard Worth Carrying Forward

The habits in this paper extend beyond COMP 330. Careers are shaped by the ability to join unfamiliar teams, understand existing systems, communicate clearly, challenge responsibly, accept feedback, and deliver work that others can trust. Technical knowledge opens the door; professional teamwork determines how much responsibility an organization can place in you.

The AI era will increase the number of artifacts and decisions moving through engineering teams. It will reward people who can reduce ambiguity, create shared context, and preserve judgment under speed. Engineers who isolate themselves with an AI assistant may produce impressive local output. Engineers who can lead human-AI teams toward coherent outcomes will shape entire systems and organizations.

Throughout my career, the strongest engineers were not simply the fastest programmers. They made everyone around them more effective. They clarified the problem, anticipated consequences, explained difficult ideas, reviewed with rigor and respect, shared credit, accepted accountability, and left the system easier to understand than they found it.

That is the standard this course asks you to practice. Do your part, but never lose sight of the whole. Use AI, but never outsource ownership. Challenge the work, but strengthen the team. Build software, but also build the shared understanding required to sustain it.

Before Your Team Begins

During the opening weeks of the course, use these questions to establish the team's operating model. The purpose is not to create bureaucracy. It is to prevent predictable ambiguity before it becomes late-semester failure.

- ✓ Do we share the same understanding of the project outcome, Cycle 1 boundary, and definition of success?
- ✓ Does every role have a primary owner, backup, and understood obligations?
- ✓ Where will decisions, work, risks, and evidence be recorded so the whole team can find them?
- ✓ How small will we keep issues and pull requests so that review remains credible?
- ✓ What response do we expect when someone identifies a defect, uncertainty, missed estimate, or AI limitation?
- ✓ How will we handle disagreement and escalation without allowing decisions to remain unresolved?
- ✓ How will every member contribute technically and understand the complete system?
- ✓ What AI tools may the team use, what must be disclosed, and what verification is required?

The COMP 330 Team Commitment

We will make work visible, keep changes reviewable, challenge claims with evidence, surface problems early, use AI responsibly, and remain collectively accountable for the system we release.

Conclusion

Software engineering teams are not effective because every member is equally skilled or because conflict disappears. They are effective because the operating model turns different knowledge and perspectives into shared, controlled, and reviewable work.

AI increases the need for that model. As production becomes faster, teams must become better at alignment, decomposition, review, evidence, and learning. The limiting capability will often be not how much work the team can generate, but how much trustworthy change it can understand and absorb.

Your COMP 330 project is an opportunity to practice the profession rather than simulate a group assignment. Build the product, but also build the team system that makes the product possible. That team system—shared intent, visible ownership, disciplined challenge, and accountable action—is one of the most durable engineering skills you can carry into your career.

References and Further Reading

[1] DORA. Capabilities: Working in Small Batches, updated 2025. https://dora.dev/capabilities/working-in-small-batches/

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

[3] 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

[4] 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

[5] Google re:Work. Team Effectiveness and Psychological Safety resources. https://rework.withgoogle.com/

[6] ACM, IEEE Computer Society, and AAAI. Computer Science Curricula 2023: Final Report. https://csed.acm.org/final-report/

[7] ACM/IEEE-CS/AAAI. CS2023 Society, Ethics, and Professionalism Knowledge Area. https://csed.acm.org/wp-content/uploads/2023/03/SEP-Version-Beta.pdf

[8] GitHub Docs. About Code Owners. https://docs.github.com/repositories/managing-your-repositorys-settings-and-features/customizing-your-repository/about-code-owners

[9] William T. O'Connell. Engineering Review and Readiness. ETIS White Paper WP-007, 2026. https://etisframework.org

[10] William T. O'Connell. Engineering Digital Colleagues. ETIS White Paper WP-010, 2026. https://etisframework.org

[11] William T. O'Connell. Engineering Education in the AI Era. ETIS White Paper WP-005, 2026. https://etisframework.org

[12] William T. O'Connell. Why Software Engineering Matters More in the AI Era. COMP-WP-001, 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.