COMP 330 PROFESSIONAL PAPER SERIES

Engineering Career Lessons

What Four Decades of Software Engineering Teach About Judgment, Growth, and Professional Trust

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-005 | Version 1.0 | Fall 2026

Core Thesis

Careers are not built by collecting technologies. They are built by repeatedly earning trust: understanding difficult problems, making sound decisions under uncertainty, communicating clearly, delivering responsibly, learning faster than conditions change, and leaving every system and team stronger than you found them.

Read this as professional advice, not as a formula. The examples come from one long career, but the principles are meant to help you shape your own.

Executive Abstract

Technology careers look linear when summarized on a résumé. They are not. They are built through incomplete information, changing markets, difficult projects, unexpected failures, strong mentors, weak assumptions, organizational pressure, and moments when an engineer must decide whether to protect appearance or protect the system. Titles and technologies are visible outcomes. The deeper career is the accumulation of judgment.

Over four decades, software engineering moved through mainframes, distributed systems, client/server computing, the Internet, enterprise platforms, cloud, data and analytics, and now artificial intelligence. Each transition rewarded people who could learn, but learning a tool was never enough. The engineers who became broadly trusted were those who could connect technology to real outcomes, explain tradeoffs, challenge weak thinking, collaborate across boundaries, and remain accountable after deployment.

The AI era intensifies these lessons. AI can accelerate implementation, research, testing, communication, and analysis. It can also create the appearance of competence faster than competence itself develops. DORA's 2025 research describes AI as an amplifier of the surrounding engineering system, while current workforce research emphasizes analytical thinking, resilience, curiosity, leadership, systems thinking, and lifelong learning alongside AI and technological literacy [1][2]. The implication for students is clear: technical capability remains essential, but durable careers will depend on how responsibly that capability is exercised.

This paper presents twelve career lessons for emerging software engineers. They concern foundations, adaptability, systems thinking, communication, evidence, failure, review, ownership, leadership, reputation, AI, and long-term growth. They are not shortcuts to a title. They are practices through which an engineer becomes the person others trust with harder problems.

Career Question

When the tools, organizations, and markets change—as they will—what professional qualities will make people continue to trust you with important work?

1. Your Career Will Not Follow the Plan You Imagine

Students often think of a career as a sequence: learn the required technologies, earn the first role, become senior, lead larger work, and advance. Real careers are less orderly. Organizations reorganize. Products fail. Markets move. Good projects are cancelled. Weak projects survive. Leaders change. A technology that appears dominant can become legacy faster than expected.

This uncertainty is not a reason to avoid planning. It is a reason to plan around durable capabilities rather than fixed predictions. You should have direction, but not rigidity. The goal is to become capable of moving toward valuable problems even when the route changes.

In my own career, each major transition required learning new platforms and methods, but the more important adjustment was learning to see the next system differently. Mainframe thinking did not simply shrink into distributed computing. Client/server systems did not simply move onto the Internet. Cloud was not merely another data center. AI is not simply faster automation. Each wave changes architecture, economics, operating models, skill expectations, and organizational power.

The lesson is to treat uncertainty as part of professional life. Build a strong base, watch the direction of the industry, and preserve enough humility to revise your plan.

2. Foundations Are What Let You Adapt

Adaptability is often described as willingness to learn new tools. That is too shallow. Real adaptability depends on foundations. An engineer who understands algorithms, state, concurrency, data, interfaces, distributed failure, security, architecture, and operating systems can learn a new framework by mapping it to durable concepts. An engineer who knows only the framework must start over when the abstraction changes.

Foundations also protect you from fluent nonsense. AI can generate a plausible explanation, implementation, or architecture almost instantly. Without a mental model of how the system should behave, you cannot distinguish a useful acceleration from a confident mistake.

ABET's current software-engineering criteria still emphasize requirements analysis, design and construction, security, verification and validation, engineering processes, and tools for complex systems [3]. The specific technologies will evolve; these responsibilities remain.

Do not mistake fundamentals for old material to finish before beginning real work. Fundamentals become more valuable as your work becomes more complex, more automated, and more consequential.

Foundation Principle

Tools increase your reach. Foundations determine whether you can judge where that reach should go.

3. Learn the Business Problem, Not Only the Technical Task

Early-career engineers are often assigned a narrow task: add an endpoint, fix a defect, migrate a component, improve a query, or implement a feature. Completing the task is necessary. Understanding why the task matters is what begins to separate an implementer from an engineer.

Every technical decision exists inside a business and operating context. There are users, costs, deadlines, risks, regulations, dependencies, support burdens, and alternatives. A technically elegant solution can be wrong if it solves the wrong problem or creates more organizational cost than value.

The best engineers ask questions that connect implementation to outcome: Who depends on this? What failure matters most? What is the cost of delay? Which constraint is real and which is inherited habit? What will operations need? What decision does this data support? What becomes harder if we choose this design?

Business understanding does not reduce technical depth. It gives technical depth direction.

4. Systems Thinking Is the Difference Between Local Success and Real Success

Many engineering failures begin with a local success. A component works, a test passes, a team meets its deadline, or a demonstration impresses a stakeholder. The larger system still fails because the change creates an interaction no one owned.

Systems thinking means understanding relationships: between requirements and architecture, components and dependencies, delivery and operations, incentives and behavior, local optimization and enterprise consequence. It means asking what happens before your component, after it, during failure, and when the original assumptions change.

This mindset becomes critical in AI-assisted engineering. A generated patch may be locally correct while weakening security, observability, maintainability, or architectural coherence. The model is not the system, and the diff is not the decision.

As your career progresses, the size of the system you are expected to understand will expand—from a function, to a service, to a product, to a portfolio, to an organization. Practice seeing beyond the immediate assignment now.

5. Communication Is an Engineering Capability

Engineering communication is not decoration added after technical work. Requirements, architecture, reviews, incident response, risk escalation, and leadership all depend on the ability to make complex reality understandable to people with different responsibilities.

Clear communication begins with clear thinking. Can you state the problem without hiding behind jargon? Can you identify the decision that must be made? Can you distinguish fact, assumption, recommendation, and uncertainty? Can you explain a tradeoff to an engineer, an executive, an operator, or a user without giving each the same presentation?

CS2023 explicitly emphasizes professional communication and professional dispositions as part of computing competence [4][5]. This is not accidental. Large systems fail when critical knowledge remains private, ambiguous, or badly translated across boundaries.

Write so another engineer can act. Speak so a decision-maker can decide. Listen closely enough to discover that the problem is different from the one you prepared to solve.

6. Evidence Builds Credibility Faster Than Confidence

Confidence can help you speak. Evidence is what makes others rely on your conclusion. A professional engineer learns to connect claims to facts, methods, tests, observations, and decisions.

When you say a change is ready, show the requirement, review, tests, limitations, and recovery path. When you recommend an architecture, explain the alternatives, constraints, and consequences. When you report a problem, separate what is known from what is suspected. When you say you learned a technology, show what you built and how you reasoned about it.

This habit matters in projects, interviews, promotions, and leadership. Vague claims such as “I worked on,” “I helped with,” or “the system is scalable” are weak because they cannot be examined. Credible professionals make bounded claims and welcome informed challenge.

Your portfolio, repository, and course work should become evidence of how you think—not merely proof that you were present.

> **Credibility Principle**
>
> Confidence says, “Trust me.” Engineering evidence says, “Here is why this conclusion deserves trust.”

7. Failure Is Valuable Only When It Changes You or the System

Every long career includes defects, missed estimates, wrong designs, poor assumptions, and decisions that look different in hindsight. Failure is not automatically educational. It becomes useful only when it produces better judgment, stronger controls, or a changed system.

The first responsibility is honesty. Do not spend energy protecting the image that everything went according to plan. Make the failure visible enough to understand. Preserve evidence. Identify contributing conditions rather than searching for one convenient person to blame.

The second responsibility is action. A postmortem that produces no changed test, design, process, ownership, or operating practice is a story, not learning. The most credible response to failure is a visible improvement in how future work is performed.

Resilience is not pretending that failure does not affect you. It is the capacity to recover, extract meaning, and continue with better judgment.

8. Review Is How Individual Work Becomes Shared Engineering

No engineer, regardless of experience, should expect important work to be accepted without challenge. Review is not an obstacle placed between the author and the result. It is the mechanism through which private reasoning becomes organizational confidence.

A strong reviewer asks whether the change solves the real problem, fits the architecture, handles failure, preserves security, can be tested, and can be operated. A strong author does not defend every line as personal identity. The objective is a better system, not victory in discussion.

AI raises the importance of review because production capacity can exceed human attention. DORA notes that AI introduces verification and integration burdens even while accelerating initial generation [1]. The answer is not superficial approval. It is smaller changes, better context, stronger automated checks, and human attention focused on consequence.

Learn to give review that is specific, respectful, and useful. Learn to receive review without becoming passive or defensive. Both are leadership skills.

9. Ownership Extends Beyond the Moment the Code Works

One of the clearest distinctions between schoolwork and professional engineering is the duration of responsibility. In a class exercise, success may end when the answer is submitted. In a real system, the change must be deployed, observed, supported, repaired, explained, and eventually replaced.

Ownership means thinking about the complete lifecycle. Who operates the system? How will a failure be detected? What is the rollback path? What dependency or assumption is most fragile? What evidence will future maintainers need? What happens when the original team is gone?

You do not need formal authority to practice ownership. You can improve the README, clarify an issue, add a missing test, document a decision, surface a risk, or leave a component easier to understand than you found it.

Reputation grows when people learn that your work does not create hidden obligations for everyone else.

Ownership Principle

Do not measure your responsibility by the lines you changed. Measure it by the consequences your change creates for users, teammates, operators, and future engineers.

10. Leadership Begins Before You Have a Title

Leadership is often confused with management authority. Engineering leadership begins whenever you improve the quality of shared decisions and shared work.

You lead when you clarify an ambiguous requirement, help a teammate understand a system, surface a risk others prefer to ignore, organize a difficult review, make a decision traceable, or remain calm during failure. You lead when you create conditions in which others can succeed.

Titles can expand your decision rights, but they cannot create credibility after the fact. People follow technical leaders because those leaders repeatedly demonstrate judgment, fairness, competence, and responsibility.

At the same time, leadership does not mean having an answer to everything. Mature leaders know when to ask, delegate, escalate, and change their mind. Intellectual humility is not weakness; it is protection against the scale of one’s own authority.

11. Your Reputation Is the Accumulation of Small Professional Choices

Careers are shaped by visible achievements, but professional trust is often built through smaller moments: whether you prepare for a meeting, respond to review, acknowledge uncertainty, meet a commitment, communicate a delay, protect a customer, credit a teammate, or admit that an assumption was wrong.

Organizations remember who reduces ambiguity and who creates it. They remember who surfaces risk early and who hides it until a crisis. They remember who can be trusted with sensitive work, who helps others grow, and who treats pressure as permission to abandon standards.

This is good news for students. You do not need a senior title to begin building a professional reputation. Every team project, repository, presentation, review, and interaction is practice.

Excellence is not one dramatic performance. It is a pattern that becomes difficult to dismiss.

12. Use AI as Leverage, Not as an Identity

AI will be part of your career, but no particular product should become your professional identity. Tools change. Vendors rise and fall. Interfaces become commodities. Your durable value is the ability to use powerful tools without surrendering understanding, judgment, or accountability.

The strongest AI-enabled engineers will know when generation is useful, when retrieval or automation is sufficient, when deterministic software is safer, and when the problem should not be automated. They will understand context, evaluation, authority, privacy, security, cost, and operational consequence.

They will also protect their own development. DORA's recent analysis warns of skill degradation and heavy verification overhead when AI use is poorly integrated [1]. Use AI to accelerate learning, compare alternatives, and extend your reach. Do not use it to avoid the struggle through which mental models are built.

AI should make you capable of taking responsibility for more—not capable of appearing productive while understanding less.

The Career Pattern Behind the Lessons

The twelve lessons reinforce one another. Technical growth without communication limits influence. Leadership without evidence becomes opinion. Adaptability without foundations becomes tool chasing. Ownership without systems thinking creates local optimization. AI without judgment creates accelerated risk.

Career stage	The visible work	The deeper professional challenge
Student / emerging engineer	Learn foundations, complete projects, contribute to teams.	Build habits of evidence, communication, review, and ownership before the stakes rise.
Early-career engineer	Implement features, fix defects, learn systems.	Connect local work to architecture, users, operations, and business purpose.
Experienced engineer	Design components, lead reviews, guide delivery.	Make sound tradeoffs, grow others, and reduce organizational ambiguity.
Technical leader / architect	Shape systems, standards, and portfolios.	Balance local and enterprise outcomes; make risk and decision rights explicit.
Executive technical leader	Set direction, allocate investment, accept consequence.	Create an engineering system in which trustworthy decisions can scale beyond individual expertise.

A Practical Model for Your Next Several Years

You do not need to master all twelve lessons at once. Career growth is iterative. Each project gives you a chance to deepen one layer while maintaining the others.

Begin by becoming dependable at bounded work. Learn the system, clarify the task, communicate progress, test honestly, and finish what you start. Then expand your scope: understand neighboring components, lead a review, improve the process, mentor a teammate, own an operational outcome, or connect a technical decision to business value.

Choose work that increases both capability and evidence. A difficult project is valuable when it leaves you with a stronger mental model, a credible result, and a story you can explain. Repetition matters. One strong performance may create an opportunity; consistent performance creates a career.

Growth horizon	Focus	Evidence of progress
Now	Foundations, professional habits, team contribution, honest AI use.	Course work, repository history, explanations, reviews, portfolio evidence.
First roles	System understanding, delivery reliability, debugging, communication.	Changes that ship safely, strong incident participation, trusted ownership.
Growing scope	Architecture, cross-team coordination, mentoring, risk judgment.	Decisions that improve multiple teams or reduce recurring failure.
Leadership	Direction, talent growth, governance, organizational design.	Stronger systems and stronger people that continue succeeding without your constant intervention.

What This Means for COMP 330

COMP 330 cannot simulate an entire career, but it can give you authentic practice in the habits that compound across one. You will work with incomplete information, collaborate with teammates, make architecture decisions, use AI, receive review, manage evidence, present claims, and improve a system after the first release.

The project matters, but the habits matter more. A feature may become obsolete. The ability to clarify intent, reason about tradeoffs, review honestly, communicate professionally, learn from failure, and own the complete result will remain useful throughout your career.

Treat the semester as a small professional engineering organization. Do not wait for an internship or full-time role to begin acting like an engineer. The professional standard starts with how you approach the work in front of you.

COMP 330 Career Objective

Leave the course with more than a completed project. Leave with a visible pattern of professional behavior that you can carry into every future engineering environment.

Personal Reflection Before the Semester Accelerates

Consider these questions privately and revisit them during the semester. They are not graded answers; they are prompts for deliberate growth.

- Which technical foundation do I most need to strengthen so that I can judge AI-generated work rather than merely accept it?
- When I work on a team, do I reduce ambiguity or add to it?

- How do I respond when review challenges work I am proud of?
- Do my repositories and explanations provide evidence of how I think?
- What kind of failure do I tend to hide, rationalize, or avoid discussing?
- What professional behavior do I want my teammates to associate with my name?
- Which responsibility do I want to be trusted with by the end of this course that I am not ready to own today?

Conclusion

A long career is not one uninterrupted rise. It is a sequence of transitions in which your tools, responsibilities, assumptions, and confidence are repeatedly tested. The engineers who continue to grow are not those who predicted every change. They are those who built durable foundations, learned quickly, communicated clearly, accepted review, owned consequences, and converted failure into better systems.

Artificial intelligence changes the pace and scale of the work, but it does not change the deepest professional requirement: other people must be able to rely on your judgment. That trust is earned slowly, through evidence and behavior, and it can be lost quickly when appearance becomes more important than responsibility.

Build your career deliberately. Learn the technology. Understand the system. Know why the work matters. Help the team. Surface the risk. Defend the decision. Own the result. Then leave the system—and the people around you—better than you found them.

Final Career Lesson

Your greatest professional achievement will not be the amount of software you produce. It will be the number of difficult situations in which people can trust you to help produce the right outcome.

References and Further Reading

[1] DORA. State of AI-assisted Software Development 2025 and related 2026 analysis of AI adoption, verification overhead, integration challenges, and skill development. https://dora.dev/research/publications/

[2] World Economic Forum. The Future of Jobs Report 2025. https://www.weforum.org/publications/the-future-of-jobs-report-2025/

[3] ABET. Criteria for Accrediting Engineering Programs, 2025–2026: Software and Similarly Named Engineering Programs. https://www.abet.org/accreditation/accreditation-criteria/criteria-for-accrediting-engineering-programs-2025-2026/

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

[5] ACM/IEEE/AAAI CS2023. Society, Ethics, and Professionalism knowledge area and professional communication guidance. https://csed.acm.org/knowledge-areas-society-ethics-and-professionalism-sep-sigcse-2022-version/

[6] William T. O'Connell. Why Software Engineering Matters More in the AI Era. COMP 330 Professional Paper COMP-WP-001, 2026. https://etisframework.org

[7] William T. O'Connell. Building a Professional Engineering Portfolio. COMP 330 Professional Paper COMP-WP-002, 2026. https://etisframework.org

[8] William T. O'Connell. Working Effectively on an Engineering Team. COMP 330 Professional Paper COMP-WP-003, 2026. https://etisframework.org

[9] William T. O'Connell. Using AI Professionally. COMP 330 Professional Paper COMP-WP-004, 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.