# Engineering Trustworthy Intelligent Systems

**ETIS Student Professional Engineering Guide**

Evidence, AI Responsibility, and Engineering Maturity

Software Engineering, Governance, and Operational Trust in the AI Era

**William T. O'Connell, Ph.D.**

FIRST EDITION EDUCATIONAL PRODUCT

# Contents

# Copyright and Use

# ETIS Educational Product Series

This document is part of the **ETIS Educational Product Series**.

The educational product series transforms *Engineering Trustworthy Intelligent Systems* from a publication into a teachable, adoptable, and stewardable engineering framework.

The series is designed for instructors, students, departments, universities, professional educators, and institutions adopting ETIS in software engineering, AI governance, responsible AI, enterprise systems, capstone, project-based, or professional-practice environments.

## Product Family

The ETIS Educational Product Series includes:

| Product | Primary Purpose |
|---|---|
| **ETIS Educational Ecosystem Guide** | Explain the full ETIS educational architecture and public product model |
| **ETIS Instructor Course Package** | Provide the instructor operating system for course design and delivery |
| **ETIS Classroom Facilitation Guide** | Help instructors run ETIS classrooms as active engineering environments |
| **ETIS Instructor Handbook** | Preserve instructor guidance, teaching judgment, and long-term stewardship practices |
| **ETIS Student Professional Engineering Guide** | Help students understand and practice professional engineering behaviors |

## Relationship to the ETIS Book

The ETIS book remains the authoritative doctrine.

The educational product series translates that doctrine into teaching, learning, adoption, classroom operation, and stewardship resources.

Educational products teach ETIS.

Adoption examples prove ETIS.

Educational stewardship sustains ETIS over time.

## Common Educational Premise

ETIS education is built on a simple premise:

> AI can produce artifacts. Engineers create trust.

Students should not merely complete assignments.

They should develop evidence of engineering maturity.

Instructors should not merely deliver content.

They should operate educational systems that help students practice trustworthy engineering.

## Shared Educational Mission

ETIS educational products teach future engineers to:

- define intent
- engineer context
- bound authority
- verify behavior
- operate reality
- explain decisions
- own outcomes

## Use of This Product

This product is intended to be used as a public educational resource.

It should be read together with the ETIS book, appendices, educational ecosystem pages, instructor resources, student resources, flagship implementation guidance, and institutional adoption guidance.

# ETIS Student Professional Engineering Guide

## Who This Is For

This guide is for students learning software engineering through ETIS.

It is intended for:

- undergraduate students
- graduate students
- capstone students
- student project teams
- early-career engineers
- students using AI-assisted development tools
- students building professional portfolio evidence

## Purpose

The ETIS Student Professional Engineering Guide helps students understand what it means to practice software engineering in the AI era.

It explains how to work like a responsible engineer, preserve evidence, use AI responsibly, participate in reviews, defend releases, and build durable professional portfolio evidence.

This is not only a guide to completing a course.

It is a guide to becoming a trustworthy engineer.

## The ETIS Student Mindset

In ETIS, the goal is not simply to build something that works.

The goal is to build something that can be:

- understood
- reviewed
- verified
- governed
- operated
- improved
- defended

A working demo is not enough.

Engineering work must leave evidence.

## Student Transformation

ETIS is designed to move students through a professional transformation.

Student ↓ Responsible Engineer ↓ Reviewer ↓ Architect ↓ Release Defender ↓ Operational Thinker ↓ Future Trustworthy Engineer

Each stage requires more accountability.

Each stage should leave evidence.

## Professional Expectations

Students are expected to:

- work responsibly in teams
- communicate clearly
- document important decisions
- preserve repository evidence
- use AI transparently
- verify AI-assisted work
- participate in reviews
- test system behavior
- disclose risks and limitations
- defend release readiness
- reflect on improvement

These expectations mirror professional engineering work.

## Repository-Centered Engineering

Your repository is not just a place to store code.

It is the system of record for your engineering work.

A strong ETIS repository may include:

docs/ requirements/ planning/ team/ architecture/ ai/ reviews/ testing/ quality/ security/ release/ operations/ observability/

src/ tests/ data/ scripts/

The repository should show what your team thought, built, verified, reviewed, and learned.

## Engineering Evidence

Evidence makes your work reviewable.

Important evidence may include:

- requirements
- assumptions
- user stories or use cases
- acceptance criteria
- architecture decisions
- work breakdown structures
- AI-use logs
- review logs
- pull request evidence
- test evidence
- defect logs
- release-readiness records
- known limitations
- operational notes
- postmortems
- final defense materials

If important work leaves no evidence, reviewers cannot evaluate it.

## AI Responsibility

ETIS does not ask you to avoid AI.

It asks you to govern AI.

You may use AI to help with:

- brainstorming
- requirements drafting
- architecture critique
- planning
- implementation
- testing ideas
- documentation
- review preparation
- defect analysis
- operational reasoning

But AI does not own your work.

You do.

Meaningful AI use should be disclosed and verified.

A useful AI-use record should explain:

- what tool was used
- what task it supported
- what output was produced
- what was accepted
- what was modified
- what was rejected
- how the result was reviewed
- how the result was verified

AI proposes.

Engineers verify.

## Team Responsibility

Team engineering requires more than dividing tasks.

Teams must coordinate:

- roles
- communication
- decisions
- repository structure
- issue tracking
- review responsibilities
- integration work
- testing
- release preparation

- final defense

A team is accountable for the whole system, not only individual contributions.

## Review Responsibility

Reviews are not obstacles.

They are how engineering work becomes stronger.

You may be asked to review:

- requirements
- architecture
- AI-use records
- pull requests
- test evidence
- release readiness
- operational risks

When reviewing, ask:

- Is the decision clear?
- Is the evidence strong?
- What risk remains?
- What assumption is hidden?
- What would fail first?
- What should be improved?

Learning to review is part of becoming an engineer.

## Release Defense

A release defense is not a presentation about how hard the team worked.

It is an evidence-backed argument that the system is ready for the claimed release state.

A strong release defense answers:

- What was built?
- What requirements does it satisfy?
- What evidence supports that claim?
- What tests were run?
- What defects remain?
- What AI assistance was used?
- How was AI-assisted work verified?
- What risks remain?
- What would need to improve before real use?

A demo may support the defense.

It does not replace the defense.

## Operational Thinking

ETIS asks students to think beyond construction.

A system must eventually operate in reality.

Operational thinking includes:

- logging
- observability
- security
- reliability
- runbooks
- supportability
- incident response
- user impact
- known limitations
- improvement planning

A system that works once is not necessarily trustworthy.

## Portfolio Thinking

Your ETIS repository can become professional portfolio evidence.

A strong repository can show future employers that you can:

- work in a team
- define requirements
- make architecture decisions
- use AI responsibly
- preserve evidence
- review work
- test behavior
- defend release readiness
- think operationally
- improve from feedback

The goal is not simply evidence of course completion.

The goal is evidence of engineering ability.

## How To Use This Resource

Use this guide throughout the course.

At the beginning, use it to understand expectations.

During the project, use it to check your team's evidence.

Before reviews, use it to prepare.

Before release defense, use it to evaluate readiness.

After the course, use it to improve your professional portfolio.

## Relationship to Other ETIS Products

This guide connects to:

- the ETIS book for doctrine

- the Educational Ecosystem Guide for educational context
- the Instructor Course Package for course design
- the Classroom Facilitation Guide for classroom operation
- the Instructor Handbook for long-term teaching stewardship

Students do not need to read every instructor product.

But they should understand that their work is part of a larger ETIS educational system.

## Bottom Line

ETIS is not asking you to avoid AI, write more documents, or follow process for its own sake.

ETIS is asking you to become the kind of engineer who can create systems that deserve trust.

That means defining intent, preserving evidence, verifying behavior, explaining decisions, governing AI use, defending releases, and owning outcomes.

That is professional engineering in the AI era.

# Part I

## Professional Engineering Mindset

## ETIS Student Starter Kit Architecture

### Purpose

This document defines the architecture of Student Starter Kits within the ETIS (Engineering Trustworthy Intelligent Systems) Educational Ecosystem.

The purpose of this architecture is to establish how ETIS translates educational theory into student engineering practice.

Student Starter Kits provide the environment where learners repeatedly practice trustworthy engineering behaviors.

They are the operational bridge between educational design and student engineering work.

### Core Philosophy

A Student Starter Kit is a product.

It is not a repository template.

It is not a collection of files.

It is not a starter assignment.

It is an intentionally assembled engineering environment.

The objective is to create environments where students naturally practice ETIS principles as part of their daily engineering activities.

The environment itself should teach engineering.

### Educational Role

Student Starter Kits occupy a specific position within the ETIS Educational Ecosystem.

ETIS Book

↓

Educational Architecture

↓

Learning Models

↓

Shared Assets

↓

Student Starter Kit

↓

Student Engineering Practice

↓

Professional Transformation

Student Starter Kits operationalize ETIS.

The completed Instructor Course Package sits alongside the Student Starter Kit.

The Instructor Course Package designs, operates, and stewards educational systems.

The Student Starter Kit provides the engineering environment where students repeatedly practice those systems.

Their relationship can be summarized as:

Instructor Course Package

↓

Student Starter Kit

↓

Student Engineering Practice

↓

Professional Transformation

The two systems are intentionally complementary.

One teaches instructors how to create trustworthy engineers.

The other provides environments where students repeatedly practice becoming trustworthy engineers.

They transform educational concepts into engineering habits.

### Core Educational Problem

Traditional software engineering courses often begin with an empty repository.

Students then gradually discover:

- Documentation
- Testing
- Architecture
- Reviews
- Traceability
- Operational thinking

These disciplines often appear late in the project.

ETIS intentionally reverses this pattern.

Students should begin inside an engineered environment that already expects professional engineering behavior.

The starter kit reduces educational randomness.

### Architectural Goals

Student Starter Kits exist to achieve several goals.

**Goal 1: Normalize Professional Engineering Structure** Students should encounter professional engineering organization immediately.

**Goal 2: Teach Repository-Centered Engineering** Repositories should function as engineering memory systems.

**Goal 3: Teach Evidence-Centered Engineering** Claims should be supported by evidence.

**Goal 4: Teach Responsible AI Collaboration** AI should be treated as a bounded collaborator.

**Goal 5: Teach Reviewability** Engineering work should be understandable by others.

**Goal 6: Teach Operational Thinking** Students should think beyond implementation.

**Goal 7: Teach Accountability** Students should own outcomes.

### Architectural Components

Every Student Starter Kit consists of three layers.

Engineering Workspace

↓

Engineering Evidence System

↓

Engineering Learning System

Each layer serves a different purpose.

### Layer 1: Engineering Workspace

This is the visible repository structure students interact with.

Examples may include:

README.md

docs/

src/

tests/

scripts/

data/

The workspace organizes engineering activities.

This layer should remain intentionally simple.

### Layer 2: Engineering Evidence System

This layer defines what evidence students are expected to preserve.

Examples include:

- Requirements
- Assumptions
- Risks
- Architecture
- Decisions
- AI-use logs
- Test evidence

- Review evidence
- Release evidence
- Operational assumptions

Students should learn that evidence accumulates continuously.

Evidence should not be generated at the end of the semester.

### Layer 3: Engineering Learning System

This layer connects repository activities to educational outcomes.

Repository activities should reinforce:

- Professional transformation
- Engineering maturity
- Review skills
- AI accountability
- Operational awareness
- Stewardship

The environment itself should help learners mature.

### The Seven Engineering Responsibilities

Every Student Starter Kit should reinforce seven ETIS responsibilities.

Define Intent

Engineer Context

Bound Authority

Verify Behavior

Operate Reality

Explain Decisions

Own Outcomes

These responsibilities should be visible throughout the engineering environment.

### Design Philosophy

Student Starter Kits should embody several design characteristics.

**Visible** Students should be able to easily understand where work belongs.

**Understandable** The environment should not overwhelm beginners.

**Reviewable** Engineering evidence should be easy to inspect.

**Sustainable** The structure should remain useful throughout the entire project lifecycle.

**AI-Aware** Responsible AI practices should be integrated naturally.

**Operationally Conscious** Students should think beyond implementation.

**Evolvable** The environment should support future growth without structural instability.

### Repository-Centered Engineering

Student Starter Kits reinforce one of the foundational ETIS concepts:

> Repositories preserve engineering memory.

Repositories should contain more than code.

They should preserve:

- Intent
- Decisions
- Evidence
- Reviews
- Risks
- Tests
- Operational assumptions
- Reflection

Students should gradually understand that engineering memory is one of their primary responsibilities.

### Evidence-Centered Engineering

Student Starter Kits should teach students that evidence is continuous.

Evidence may include:

Requirements

↓

Architecture

↓

Implementation

↓

Tests

↓

Reviews

↓

Release Evidence

↓

Operational Learning

Every phase should leave evidence behind.

### AI-Aware Engineering

Student Starter Kits should support responsible AI usage.

Students should have clear places to preserve:

- AI policies

- AI-use logs
- Verification notes
- Human review evidence

The starter kit should reinforce:

> AI proposes; engineers verify.

AI should not become an invisible participant.

### Relationship to Learning Models

Student Starter Kits consume learning models.

Learning models explain how students develop.

Starter kits provide environments where that development occurs.

Examples include:

- Professional Transformation Model
- Engineering Maturity Model
- Software Engineering Learning Progression
- Two-Cycle Engineering Model

The starter kit operationalizes those models.

### Relationship to Shared Assets

Student Starter Kits consume shared assets.

Shared assets remain authoritative.

Examples include:

shared ETIS educational assets

playbooks/

templates/

workflows/

ai_engineering/

assessment/

Student Starter Kits may surface these assets in student-friendly ways.

They should not permanently duplicate them.

### Relationship to Adoption Examples

Adoption examples explain how starter kits are used.

The current flagship implementation is:

adoption_examples/ ‘— loyola_comp330/

The adoption example explains instructional use.

The starter kit provides the student environment.

The relationship is complementary.

The flagship implementation proves the doctrine.

It does not become the doctrine.

The COMP330 starter kit should never become the universal ETIS starter kit.

Future educational implementations should inherit ETIS principles while adapting to their own environments.

Institutions should inherit ETIS doctrine, not ETIS implementations.

### Architectural Boundaries

Student Starter Kits do not own:

- Course schedules
- Syllabi
- Rubrics
- Instructor slides
- Instructor analysis tools
- Educational architecture
- Learning models
- Shared asset sources

Student Starter Kits own the student engineering environment.

### Growth Model

New starter kits should only be created when educational environments materially differ.

Examples may include:

- Undergraduate software engineering
- Graduate software engineering
- Capstone programs
- Professional workshops
- AI governance programs

Different semesters alone should not create new starter kits.

### Governance Rules

The following rules should remain stable.

**Starter kits are products.**

**Starter kits teach through structure.**

**Starter kits should consume shared assets.**

**Starter kits should preserve provenance.**

**Starter kits should avoid unnecessary complexity.**

**Starter kits should evolve from educational evidence.**

**Starter kits should support long-term maintainability.** These rules help preserve architectural consistency.

### Layer Separation Rules

The Student Starter Kit intentionally separates engineering activities into distinct layers.

docs/ → Think

src/ → Build

tests/ → Verify

data/ → Support

scripts/ → Automate

These responsibilities should remain distinct.

Do not duplicate engineering evidence across layers.

Do not move educational artifacts into implementation layers.

Do not allow tooling implementations to become educational doctrine.

This separation helps preserve long-term maintainability.

### Future Evolution

Future enhancements may include:

- Multiple starter kit variants
- Discipline-specific starter kits
- Professional training starter kits
- AI governance starter kits

Growth should occur intentionally.

The architecture should remain simple.

### Core Thesis

Student Starter Kits are where ETIS becomes practice.

They provide environments where students repeatedly exercise the habits of trustworthy engineering.

The goal is not to provide files.

The goal is to create engineering environments that help learners define intent, engineer context, bound authority, verify behavior, operate reality, explain decisions, and own outcomes.

That repeated practice is what ultimately produces trustworthy engineers.

## Starter Kit Design Principles

### Purpose

This document defines the design principles that govern Student Starter Kits within the ETIS (Engineering Trustworthy Intelligent Systems) Educational Ecosystem.

The purpose of these principles is to ensure that Student Starter Kits remain intentional, understandable, maintainable, and aligned with trustworthy engineering practices.

These principles should guide the creation of all current and future starter kits.

They are intended to create consistency without preventing thoughtful adaptation.

### Core Philosophy

Student Starter Kits should teach engineering through environment design.

The environment itself should reinforce trustworthy engineering behaviors.

Students should learn engineering practices naturally by working inside well-designed systems rather than being introduced to those practices as disconnected requirements throughout a course.

The environment should become an instructor.

The environment should teach without overwhelming.

Students should repeatedly encounter trustworthy engineering behaviors without becoming burdened by unnecessary complexity.

Well-designed environments reduce educational friction while preserving engineering accountability.

### Principle 1: Design for Professional Practice

Student Starter Kits should resemble professional engineering environments.

Students should encounter:

- Repository organization
- Documentation expectations
- Review expectations
- Evidence expectations
- Operational awareness

Students should not begin inside artificial academic environments that bear little resemblance to professional engineering practice.

The objective is not realism for its own sake.

The objective is to normalize responsible engineering behavior.

### Principle 2: Design for Repository-Centered Engineering

Repositories should function as engineering memory systems.

Starter kits should make it obvious that repositories contain more than code.

Students should have visible places to preserve:

- Intent
- Context
- Decisions

- Risks
- Reviews
- Tests
- AI usage
- Release evidence
- Operational knowledge

Repository organization should support long-term understanding.

### Principle 3: Design for Evidence-Centered Engineering

Starter kits should teach that engineering claims require evidence.

Students should be encouraged to answer questions such as:

- What was required?
- What was built?
- What was tested?
- What was reviewed?
- What risks remain?
- What limitations exist?

The repository should make evidence creation a normal activity.

Evidence should accumulate continuously.

### Principle 4: Design for Understandability

Students should be able to quickly understand the engineering environment.

Starter kits should avoid unnecessary complexity.

The organization should be intuitive.

Students should spend their effort solving engineering problems rather than deciphering repository structure.

Complexity should be introduced intentionally and only when educationally valuable.

The governing principle should be:

> Scale complexity, not accountability.

As students mature, engineering challenges may become more sophisticated.

Engineering accountability should remain visible from the beginning.

### Principle 5: Design for Progressive Maturity

Starter kits should support learner growth.

The environment should remain useful from project launch through release readiness and reflection.

The same repository should support progression from:

Task Completion ↓ Evidence-Based Engineering ↓ Reviewable Engineering ↓ Operational Thinking

The environment should mature with the learners.

### Principle 6: Design for Reviewability

Engineering work should be easy to inspect.

A reviewer should be able to understand:

- What problem is being solved
- What decisions were made
- What changed
- What was reviewed
- What was tested
- What risks remain

Starter kits should make review a natural activity rather than a special event.

### Principle 7: Design for Responsible AI Collaboration

AI is an expected participant in modern engineering environments.

Starter kits should support responsible AI use.

Students should have places to preserve:

- AI policies
- AI-use logs
- Verification notes
- Human review evidence

Starter kits should reinforce:

> AI proposes; engineers verify.

AI should never become invisible.

### Principle 8: Design for Operational Thinking

Students should think beyond implementation.

Starter kits should encourage learners to ask:

- What happens after release?
- How would someone operate this?
- How would failures be discovered?
- What would future maintainers need to know?

Operational awareness should appear early rather than late.

### Principle 9: Design for Accountability

Students should own outcomes.

Starter kits should encourage students to preserve:

- Known limitations
- Risks
- Assumptions
- Decisions
- Release evidence

Students should understand that engineering accountability extends beyond implementation.

### Principle 10: Design for Long-Term Maintainability

Starter kits should remain stable over time.

The structure should evolve carefully.

New folders, documents, and expectations should be introduced only when they improve educational outcomes.

Avoid complexity for its own sake.

The environment should remain sustainable for both students and instructors.

### Principle 11: Design for Reuse

Starter kits should consume reusable ETIS assets rather than duplicate them.

Starter kits should rely upon:

shared ETIS educational assets

Examples include:

- Playbooks
- Templates
- Workflows
- AI Engineering assets
- Assessment assets

The starter kit assembles these assets into a student experience.

It should not become another repository for educational intellectual property.

### Principle 12: Design for Provenance

Starter kits should preserve their educational origins.

Students and future adopters should understand:

- Where the starter kit originated
- Which educational implementation uses it
- How it relates to ETIS

Provenance preserves educational memory.

### Principle 13: Design for Evolution Through Evidence

Starter kits should evolve based on observed educational outcomes.

Useful inputs include:

- Student difficulties
- Repository analysis
- Instructor observations
- Assessment trends
- Reflection artifacts

Starter kits should not change based solely on preference.

Changes should be evidence informed.

### Anti-Patterns

The following anti-patterns should be avoided.

**Empty Repository Syndrome** Students begin with no engineering structure.

**Documentation Dumping** Students generate documentation only at the end.

**File Proliferation** Too many folders or documents overwhelm students.

**Hidden Expectations** Students are expected to produce evidence without knowing where it belongs.

**Invisible AI** AI usage occurs without disclosure or verification.

**Demo Thinking** Students equate working software with engineering completion.

**Educational Drift** Starter kits slowly accumulate unrelated materials over time.

These anti-patterns reduce educational effectiveness.

### Relationship to the Educational Ecosystem

These principles support all layers of ETIS.

ETIS Book ↓ Educational Architecture ↓ Learning Models ↓ Shared Assets ↓ Student Starter Kit ↓ Student Practice

The principles help translate ETIS into repeatable engineering behavior.

### Future Evolution

These principles should remain relatively stable.

Future starter kits may vary in structure, audience, and complexity.

However, the underlying design philosophy should remain consistent.

Stability helps create a recognizable ETIS educational experience across implementations.

### Core Thesis

Student Starter Kits should be intentionally engineered environments that teach trustworthy engineering through repeated practice.

The goal is not to provide students with files.

The goal is to provide environments that naturally reinforce professional engineering behaviors every time students interact with the repository.

## Starter Kit Evolution Model

### Purpose

This document defines how Student Starter Kits evolve within the ETIS (Engineering Trustworthy Intelligent Systems) Educational Ecosystem.

The purpose of this model is to ensure that Student Starter Kits improve over time while remaining understandable, stable, and maintainable.

Student Starter Kits are educational products.

Like all products, they should evolve intentionally.

However, intentional evolution is different from uncontrolled accumulation.

This model exists to preserve that distinction.

### Core Philosophy

Student Starter Kits should evolve from evidence rather than preference.

Changes should be driven by observed educational outcomes rather than convenience, trends, or individual instructor preferences.

Starter Kits should become better over time without becoming larger, more complex, or more confusing.

The objective is sustainable improvement.

### Core Principle

Student Starter Kits should become more effective, not more complicated.

Complexity is not a measure of educational maturity.

The most mature starter kit is often the one that teaches the most with the least unnecessary structure.

Every addition should justify its existence.

### Evolution Objectives

Starter Kit evolution should support five objectives.

**Improve educational effectiveness** The starter kit should better support student learning over time.

**Improve engineering practice** The starter kit should better reinforce trustworthy engineering behaviors.

**Improve understandability** The starter kit should become easier for students to navigate and understand.

**Improve maintainability** The starter kit should remain sustainable for instructors and adopters.

**Improve reuse** The starter kit should become easier to consume across implementations.

### Sources of Evolution

Starter Kit improvements should originate from observable evidence.

Appropriate sources include:

**Student difficulties** Patterns of student confusion may reveal architectural weaknesses.

**Repository analysis** Instructor analysis tools may reveal structural gaps.

**Assessment outcomes** Assessment results may reveal areas needing reinforcement.

**Instructor observations** Repeated classroom observations may indicate improvement opportunities.

**Student reflections** Student feedback may identify friction points.

**Implementation experience** Long-term usage may identify simplifications or improvements.

These inputs should drive evolution.

### Evolution Lifecycle

Starter Kit changes should follow a deliberate process.

Observation

↓

Pattern Identification

↓

Improvement Proposal

↓

Educational Review

↓

Starter Kit Update

↓

Implementation Evaluation

↓

Long-Term Adoption

Educational reviews should evaluate impacts across the larger ETIS ecosystem before changes are accepted.

Changes should be evaluated against:

- student experiences,
- instructor experiences,
- adoption examples,
- educational reuse,
- and long-term maintainability.

Starter Kit evolution should strengthen ecosystem coherence rather than create isolated improvements.

Educational systems should become more integrated over time, not more fragmented.

The goal is deliberate improvement rather than reactive change.

### What Should Trigger Change

Starter Kit changes should occur when:

- Students repeatedly struggle with the same concepts.
- Important engineering evidence is consistently missing.
- Students misunderstand repository purpose.
- AI accountability is insufficient.
- Reviewability is difficult.
- Operational thinking is underrepresented.
- Educational complexity can be reduced.

These are legitimate reasons to evolve the product.

### What Should Not Trigger Change

Starter Kit changes should not occur solely because:

- A new semester begins.
- A new instructor teaches the course.
- New tools become popular.
- A new AI model is released.
- Additional documents seem useful.
- Individual preferences change.

Change should be evidence informed.

### Stable Components

Certain Starter Kit concepts should remain relatively stable.

These include:

**Repository-centered engineering** Repositories preserve engineering memory.

**Evidence-centered engineering** Claims require evidence.

**Responsible AI collaboration** AI remains bounded and accountable.

**Reviewability** Engineering work should be understandable by others.

**Operational awareness** Students should think beyond implementation.

**Accountability** Students own outcomes.

These concepts should not fluctuate between semesters.

### Flexible Components

Some areas may evolve more frequently.

Examples include:

- Specific templates
- Classroom exercises
- Example artifacts

- Documentation examples
- Tooling integrations
- AI guidance examples

These components may adapt while preserving the underlying philosophy.

### Evolution Through Simplification

Improvement often means removing things.

Starter Kits should periodically ask:

- Is this folder necessary?
- Is this document being used?
- Is this expectation clear?
- Is this duplicating another asset?
- Is this educationally valuable?

Simplification is a legitimate form of evolution.

Removing unnecessary complexity often improves learning.

### Relationship to Shared Assets

Starter Kits should not absorb reusable educational intellectual property.

If a Starter Kit repeatedly introduces reusable materials, those materials should be evaluated for promotion into:

shared ETIS educational assets

The Starter Kit should remain an assembled environment rather than a growing content repository.

### Relationship to Adoption Examples

Adoption examples help inform Starter Kit evolution.

Educational implementations provide valuable evidence about:

- Student behavior
- Engineering challenges
- Educational friction points
- Missing structures

Implementations become feedback mechanisms for product improvement.

### Relationship to Instructor Analysis

Instructor analysis tools may become important inputs.

Examples include:

- Repository analysis scripts
- Pull request analysis
- Documentation analysis
- Evidence completeness analysis

These tools help identify patterns.

However, they should inform decisions rather than automatically drive changes.

Human educational judgment remains essential.

### Evolution Maturity Stages

Starter Kits themselves may mature over time.

**Stage 1: Foundational** Basic repository structure exists.

**Stage 2: Structured** Engineering evidence expectations become visible.

**Stage 3: Integrated** AI, reviews, testing, and release evidence are incorporated.

**Stage 4: Optimized** Educational friction is reduced.

**Stage 5: Sustainable** The Starter Kit is stable, reusable, and maintainable.

These stages describe product maturity rather than student maturity.

### Anti-Patterns

The following anti-patterns should be actively avoided.

**Semester Bloat** Every semester adds new folders and files.

**Tool Chasing** The structure changes whenever new technologies appear.

**Asset Hoarding** Reusable materials accumulate inside the Starter Kit.

**Complexity Creep** The environment becomes difficult to understand.

More files does not necessarily produce better engineers.

**Instructor Personalization** The Starter Kit becomes dependent on one instructor's preferences.

**Educational Drift** The Starter Kit slowly diverges from ETIS principles.

These anti-patterns reduce long-term sustainability.

### Stewardship Responsibilities

Starter Kit stewards should periodically evaluate:

- Is the structure still understandable?
- Is engineering evidence visible?
- Is AI accountability sufficient?
- Is repository-centered engineering being reinforced?
- Are students learning the intended behaviors?
- Is complexity increasing unnecessarily?

These questions help maintain product quality.

### Future Evolution

Student Starter Kits will likely expand over time.

Future variants may support:

- Graduate software engineering
- Capstone programs
- Professional workshops
- AI governance training
- Organizational learning

New variants should emerge only when educational needs materially differ.

The architecture should remain stable even as implementations grow.

### Core Thesis

Student Starter Kits should evolve like well-engineered products.

They should become better because educational evidence demonstrates improvement opportunities.

They should not become larger simply because time passes.

The ultimate goal is to create stable engineering environments that consistently help students develop trustworthy engineering habits.

# Part II

## Responsible AI and Engineering Workflow

## AI-Assisted Engineering Guidance

**Classification:** ETIS Educational Ecosystem Asset
**Category:** AI Engineering
**Origin:** Loyola University Chicago COMP 330 — Software Engineering
**Implementation Provenance:** COMP330 Flagship Implementation
**Authoritative Format:** Markdown
**Purpose:** Student and team guidance for responsible AI-assisted engineering across the software development lifecycle

### Purpose

This guidance explains how students and engineering teams should use AI tools responsibly during ETIS-aligned software engineering work.

The original version of this asset was created for Loyola University Chicago COMP 330 — Software Engineering. It has been normalized as an ETIS AI Engineering shared asset while preserving COMP330 provenance.

The goal is not to restrict useful tools.

The goal is to make AI-assisted engineering visible, reviewable, verifiable, and accountable.

AI tools may help teams think, draft, code, review, test, document, and prepare releases. However, AI assistance must be disclosed, reviewed, verified, and owned by the team.

Students must be able to explain every submitted artifact, every accepted decision, and every line of submitted code.

### Core Policy

AI tools are encouraged throughout the software development lifecycle.

AI may assist with:

- Requirements
- Planning
- Architecture
- Implementation
- Testing
- Review
- Documentation
- Release preparation
- Reflection
- Operational analysis

However, AI-generated or AI-assisted output is proposed engineering material until humans review, validate, govern, and intentionally accept it.

The engineering team owns the outcome.

### Why This Guidance Exists

AI can accelerate engineering work.

It can also introduce hidden assumptions, hallucinated APIs, weak tests, insecure code, unexplained dependencies, poor architecture, and false confidence.

Professional teams use AI with discipline.

AI should improve engineering practice, not obscure responsibility.

Professional engineering evidence exists to support review, validation, governance, and operational understanding. It should not be used merely to create the appearance of maturity.

### Student Policy Summary

**AI use is encouraged.** Students may use AI across requirements, planning, architecture, implementation, testing, review, documentation, release preparation, and reflection.

**Disclosure is required.** Teams must record meaningful AI assistance in the AI-use log and in pull requests or artifact notes where appropriate.

**Human review is required.** AI-generated output is proposed engineering material until humans review, validate, govern, and intentionally accept it.

**The team owns the outcome.** The team is responsible for correctness, security, architecture fit, maintainability, testing, governance, and operational impact.

**Explanation is required.** Students must be able to explain the work they submit.

If the team cannot explain it, the team should not submit it.

**Evidence is required.** Repository evidence should remain organized and discoverable through the README, repository structure, pull requests, linked issues, and related engineering documentation.

### Allowed, Expected, and Not Acceptable

| Category | Meaning |
|---|---|
| Allowed and encouraged | Using AI to brainstorm requirements, find edge cases, critique plans, draft tests, explain errors, suggest refactoring, review code, draft documentation, or prepare release notes. |
| Expected professional behavior | Disclose meaningful AI use, review outputs carefully, verify with tests or inspection, reject weak suggestions, and record what the team accepted, modified, or rejected. |
| Not acceptable | Submitting AI-generated work the team cannot explain; hiding AI use; pasting full generated systems into main without review; using AI to bypass learning; exposing private data, credentials, real student records, or sensitive university information. |

### What Must Be Disclosed

Students do not need to log every spelling correction, grammar edit, or tiny autocomplete.

Teams should disclose meaningful AI assistance that affects engineering content, project decisions, code, tests, architecture, review, release evidence, or presentation claims.

Meaningful AI assistance includes:

- Generated or substantially revised code, tests, configuration, scripts, diagrams, documentation, or presentation content
- AI-assisted requirements, user stories, acceptance criteria, risks, estimates, scope decisions, or architecture decisions
- AI review of pull requests, security concerns, defects, test gaps, design tradeoffs, or release readiness
- AI-generated suggestions the team rejected when rejection materially affected scope, architecture, risk, or release decisions
- AI assistance used to interpret project evidence, summarize work, prepare release notes, or shape the final presentation

The test is simple:

Did AI materially affect the engineering work, evidence, decision, or claim?

If yes, disclose it.

### Minimum AI-Use Log Entry

The AI-use log does not need to be long.

It needs to be honest, inspectable, and useful.

A typical location is:

/docs/ai/ai-use-log.md

| Field | What to Record |
|---|---|
| Date | When the AI assistance occurred. |
| Tool | Tool used, such as GitHub Copilot, ChatGPT, Gemini Code Assist, Gemini CLI, Claude, Microsoft Copilot, or another approved tool. |
| Task | What the team asked AI to help with. |
| Artifact or location | Repository path, issue, pull request, test, or document affected. |
| Accepted / modified / rejected | What the team did with the output. |
| Verification | How the team checked it: review, test, comparison to requirements, security check, manual run, or team explanation. |
| Owner | Team member responsible for ensuring the entry is accurate. |

### AI Use Across the Software Development Lifecycle

| Lifecycle Area | Good AI Uses | Required Control |
|---|---|---|
| Requirements | Ask for missing stakeholders, edge cases, weak acceptance criteria, ambiguity, assumptions, and out-of-scope risks. | Do not let AI expand the project beyond approved scope. |

| Lifecycle Area | Good AI Uses | Required Control |
|---|---|---|
| Planning | Use AI to identify task breakdowns, risks, dependencies, estimate assumptions, and schedule threats. | AI estimates are not truth. Teams must adjust based on actual capacity and evidence. |
| Architecture | Use AI to critique boundaries, coupling, data ownership, API contracts, failure paths, permissions, and governance points. | Do not accept architecture the team cannot explain or maintain. |
| Construction | Use AI for scaffolding, code suggestions, debugging, refactoring ideas, and explaining unfamiliar APIs. | Generated code must be reviewed, tested, and connected to issues and pull requests. |
| Review | Use AI as an additional reviewer for correctness, maintainability, security, architecture fit, tests, and dependencies. | AI review does not replace human review. |
| Testing | Use AI to propose test cases, negative paths, edge cases, regression tests, and failure scenarios. | AI-generated tests often mirror assumptions; teams must challenge the implementation. |
| Operations / Observability | Use AI to help analyze logs, identify operational anomalies, summarize incidents, draft runbooks, propose recovery steps, or critique observability gaps. | AI operational suggestions must be validated against actual runtime evidence, recovery procedures, governance expectations, and release constraints. |
| Release and Presentation | Use AI to draft release notes, summarize changes, prepare demo scripts, and check clarity. | Release claims must be supported by repository evidence, not polished wording. |

### GitHub-First AI Workflow Expectations

ETIS-aligned student teams should use AI within a repository-centered engineering workflow.

Teams should:

- Use issues to define the work before asking AI to produce or revise artifacts.
- Use branches and pull requests so AI-assisted changes are visible and reviewable.
- Explain in pull requests why the AI-assisted change exists.
- Identify what evidence supports the change.
- Identify what risks were considered.
- Disclose meaningful AI assistance in the pull request description or linked AI-use log entry.
- Use AI review as an additional signal, not as the engineering approval authority or merge decision maker.
- Update requirements, tests, architecture notes, decisions, or release evidence when AI-assisted changes affect them.
- Maintain release or version evidence identifying the repository state associated with major demon-

strations, presentations, and operational reviews.

A useful rule for student teams is:

No issue, no branch, no pull request, no review, no merge.

### Privacy, Security, and Data Rules

Students and teams must not paste the following into AI tools:

- Real student records
- Grades
- Private university data
- Credentials
- Tokens
- API keys
- Passwords
- Personal information
- Confidential material
- Sensitive operational information

Use synthetic sample data unless the instructor explicitly approves another approach.

Do not connect AI tools to live production systems or real institutional workflows without instructor approval.

If an AI tool suggests a package, API, database, or integration, document the dependency and review license, security, and maintainability risks.

If AI-generated content affects user-facing behavior or system actions, document the human approval point and what evidence is logged.

### Tool Guidance

Tool availability, limits, and institutional access change quickly.

Students should use free or university-accessible tools when possible and should not pay for tools unless they personally choose to do so.

GitHub-centered tools are preferred when they fit the repository, issue, pull request, and evidence workflow used in the course.

Possible tool categories include:

| SDLC Slice | Suggested Tool Types | Course Use | Student Note |
|---|---|---|---|
| GitHub workflow | GitHub Copilot, GitHub code review features, repository assistants | Coding assistance, chat, command-line help, code review, pull request support, repository-centered development | Use within issues, branches, pull requests, and review discipline. |
| Coding in editor | GitHub Copilot in Visual Studio Code, Gemini Code Assist, or similar IDE tools | Inline suggestions, code explanation, debugging help, test ideas, and code generation | Do not accept code blindly. |

| SDLC Slice | Suggested Tool Types | Course Use | Student Note |
|---|---|---|---|
| Command-line / local workflow | GitHub Copilot CLI, Gemini CLI, or similar terminal tools | Explaining commands, generating scripts, exploring local files, debugging errors | Read commands before running them. Never paste secrets. |
| Requirements / planning / documentation | ChatGPT, Microsoft Copilot, Gemini, Claude, or similar tools | Brainstorming, critique, rewriting, finding gaps, summarizing decisions, drafting documentation | Treat output as draft analysis. Verify against project scope. |
| Team collaboration | GitHub integrations, Teams integrations, notification tools | Repository notifications, issue/PR visibility, discussion-to-action workflow | Optional. GitHub remains the authoritative engineering record. |
| Security / dependency awareness | Dependabot, code scanning, Snyk, SonarQube, or similar tools | Dependency vulnerability awareness, static analysis, code quality, security review | Tools identify risks. Teams must classify, mitigate, document, and explain them. |
| Observability / operations | GitHub Actions logs, logging tools, monitoring tools, observability assistants | Runtime diagnostics, operational troubleshooting, release investigation | Operational evidence must remain explainable and reviewable. |

This guidance intentionally avoids depending on one vendor or tool because AI tooling changes rapidly.

Use tools that support engineering discipline.

Do not let tools replace engineering discipline.

### Quick Student Checklist

Before accepting AI-assisted work, teams should ask:

- Did AI materially help produce or revise this artifact?
- Did the team disclose the assistance in the AI-use log or pull request?
- Can the team explain the output without reading the AI transcript?
- Did the team verify the result with review, tests, inspection, or evidence?
- Did the output stay within approved project scope?
- Did the team avoid private data, credentials, and unsafe integrations?
- Is the final artifact stored in GitHub where a reviewer can inspect it?
- Could another engineering team inspect the repository evidence and understand the AI-assisted engineering decisions without interviewing the original team?

### Reusable Team AI-Use Statement

Teams may adapt the following statement for their repository.

This project may use AI tools for requirements analysis, planning, architecture critique, implementation assistance, testing ideas, review support, documentation, and presentation preparation. All AI-assisted output is reviewed, modified when needed, verified by the team, and disclosed in the project AI-use log.

The team remains responsible for the correctness, security, maintainability, explainability, governance, operational behavior, and release readiness of all submitted work.

**Core Thesis**

AI use is expected.

Blind trust is not.

Professional engineering teams use AI to accelerate engineering work while preserving human judgment, verification, review discipline, governance, operational accountability, and ownership.

The responsible question is never:

Did AI generate this?

The responsible question is:

Did humans review, verify, understand, disclose, and accept responsibility for this?

## AI Use and Disclosure Expectations Guide

**Classification:** ETIS Educational Ecosystem Asset
**Category:** AI Engineering
**Origin:** Loyola University Chicago COMP 330 — Software Engineering
**Implementation Provenance:** COMP330 Flagship Implementation
**Authoritative Format:** Markdown
**Purpose:** Student-facing expectations for meaningful AI-use disclosure and verification

### Purpose

This guide defines expectations for AI use and disclosure in ETIS-aligned software engineering work.

The original version of this asset was created for Loyola University Chicago COMP 330 — Software Engineering. It has been normalized as an ETIS AI Engineering shared asset while preserving COMP330 provenance.

AI tools are encouraged because professional software engineering is now AI-assisted.

The expectation is not to avoid AI.

The expectation is to use AI responsibly, disclose meaningful use, verify results, and own the final engineering work.

### Core Principle

AI may help produce requirements, plans, code, tests, documentation, reviews, release notes, and presentation materials.

The team still owns:

- Correctness
- Architecture fit
- Maintainability
- Security
- Testing
- Governance
- Operational impact
- Release readiness

AI generated it.

is never an engineering defense.

The professional question is whether the result was reviewed, verified, understood, and responsibly accepted.

AI-generated output is proposed engineering material until humans review, validate, govern, and intentionally accept it into the operational system.

### Appropriate AI Use

AI may be used to support engineering work throughout the software development lifecycle.

Appropriate uses include:

- Brainstorming requirements, user stories, acceptance criteria, risks, edge cases, and test ideas
- Reviewing artifacts for ambiguity, missing assumptions, weak traceability, or unclear ownership
- Drafting or improving code when the team reviews, tests, and understands the result

- Generating candidate tests, CI/CD workflow drafts, documentation, README content, release notes, and demo scripts
- Assisting with observability ideas, operational diagnostics, runbooks, postmortems, release-readiness reviews, and governance analysis when the team validates the results
- Using AI as a reviewer, critic, tutor, or second set of eyes during engineering work

The standard is not whether AI was used.

The standard is whether the team used AI professionally.

### Unacceptable or Unprofessional AI Use

The following are unacceptable:

- Submitting AI-generated work the team cannot explain, defend, test, or maintain
- Using AI to create the appearance of engineering maturity without corresponding review, validation, testing, or operational evidence
- Merging AI-generated code without human review, testing, and architecture-fit judgment
- Inventing evidence, fake test results, fake citations, fake review comments, or fake repository history
- Using AI to hide defects, inflate progress, obscure contribution, or make unsupported claims
- Entering secrets, credentials, private data, real student data, or sensitive information into AI tools without explicit approval

AI may support learning.

AI may not replace learning.

### What Must Be Disclosed

Teams should disclose AI use when it materially helped create, review, modify, or decide project work.

Disclosure does not need to list every minor grammar edit.

Disclosure must capture meaningful engineering assistance.

Meaningful AI-use evidence should normally be captured inside the repository using:

- AI-use logs
- Pull requests
- Review artifacts
- Architecture decisions
- Release evidence
- Related engineering documentation

### Disclosure Categories

| Category | Examples that should be disclosed |
|---|---|
| Requirements / planning | AI helped draft requirements, identify missing assumptions, generate acceptance criteria, estimate tasks, or analyze risk. |
| Architecture / design | AI proposed components, interfaces, data flow, governance boundaries, tradeoffs, or ADR language. |

| Category | Examples that should be disclosed |
|---|---|
| Implementation | AI generated, refactored, debugged, or substantially modified code, configuration, scripts, or CI/CD files. |
| Testing / validation | AI generated tests, edge cases, adversarial cases, golden cases, test plans, or validation rubrics. |
| Review / security | AI assisted with code review, dependency review, security concerns, defect analysis, or release readiness checks. |
| Documentation / release | AI drafted or revised README content, release notes, runbooks, known limitations, presentation materials, or demo scripts. |

### Minimum AI-Use Log Entry

For meaningful AI assistance, the AI-use log should identify:

- Date
- Tool used
- Task or artifact affected
- What AI contributed
- What the team accepted, modified, or rejected
- How the result was verified
- Owner responsible for the final decision

AI-related engineering changes should also remain traceable through repository commits, pull requests, reviews, tests, and supporting evidence when applicable.

A typical AI-use log location is:

/docs/ai/ai-use-log.md

### Good and Weak Disclosure Examples

| Disclosure Quality | Example |
|---|---|
| Good | Used AI to generate initial negative test ideas for REQ-04. Team selected three cases, rewrote them, added two tests, and verified they fail when the authorization rule is removed. |
| Good | Used AI to critique the architecture diagram for missing boundaries. Team accepted the concern about data ownership, updated architecture.md, and documented the decision in ADR-0003. |
| Weak | Used ChatGPT for coding. |
| Weak | AI helped with documentation. |
| Weak | AI generated the architecture. |

Good disclosure explains what AI helped with, what humans did with it, and how the team verified the final result.

Weak disclosure hides engineering responsibility behind vague wording.

### Verification Expectations

AI-assisted work must be verified before acceptance.

**Code** Code must be reviewed through pull requests, engineering review, and human validation before merge, especially if AI generated or significantly modified it.

**Architecture and design** AI-generated architecture or design suggestions should be evaluated for architectural consistency, maintainability, operational supportability, governance, and long-term engineering impact.

**Testing** Tests should prove important behavior, failure paths, and acceptance criteria rather than only the happy path.

AI-generated tests must be inspected because they may mirror assumptions or reinforce implementation mistakes.

**Security and governance** Security, data handling, dependency, permission, and governance concerns must be inspected when AI touches implementation or workflow behavior.

**Documentation** Documentation created with AI must be accurate, current, and consistent with the repository.

**Release claims** AI-assisted release notes, summaries, or presentations must be supported by repository evidence.

Polished wording is not release evidence.

### Sensitive Data Rule

Do not place any of the following into AI tools without explicit approval:

- Secrets
- Passwords
- Tokens
- Credentials
- Private user data
- Real student data
- Confidential information
- Unnecessary sensitive content

Use synthetic data unless explicitly approved.

When in doubt, remove, mask, or synthesize the data before using AI assistance.

### Team Accountability Standard

The team may use AI throughout the software development lifecycle, but the team must be able to explain the submitted work.

The team must be able to point to evidence showing:

- What AI helped produce
- What humans accepted
- What humans modified

- What humans rejected
- How the result was verified
- Who owned the final decision

The team is responsible for defects, security issues, poor design choices, missing tests, and unsupported claims even when AI contributed.

### Pull Request Disclosure Guidance

When AI materially affects a change, the pull request should identify:

- What AI helped produce or review
- Which files or artifacts were affected
- What the team changed after AI assistance
- How the result was tested or reviewed
- Any remaining risks or limitations

Example:

AI was used to suggest initial validation logic and negative test cases for REQ-04. The team modified the generated code, added three tests, reviewed the authorization boundary manually, and verified the tests fail when the authorization rule is removed.

This creates reviewable engineering evidence.

### Student Reflection Guidance

Students may also use AI-use evidence in reflection.

Useful reflection questions include:

- Where did AI improve our engineering process?
- Where did AI create risk or confusion?
- What AI suggestions did we reject?
- How did we verify accepted AI output?
- Did AI change our architecture, scope, tests, or release claims?
- Could another team understand our AI-assisted decisions from repository evidence?

Reflection should connect AI use to engineering maturity.

### Bottom Line

AI use is expected.

Blind trust is not.

Professional engineering teams use AI to accelerate engineering work while preserving human judgment, verification, review discipline, governance, operational accountability, and ownership.

AI should make engineering more reviewable, not less.

### Core Thesis

AI assistance does not reduce engineering responsibility.

It increases the need for evidence, disclosure, review, verification, and accountability.

A team may use AI throughout the project, but the team must still be able to explain, defend, test, maintain, and own the final engineering work.

## Engineering Repository Structure Guide

**Classification:** ETIS Educational Ecosystem Asset
**Category:** Workflow
**Origin:** Loyola University Chicago COMP 330 — Software Engineering
**Authoritative Format:** Markdown
**Typical Repository Location:** Repository root and `/docs/`

### Purpose

This guide explains the standard engineering repository structure used in an ETIS-aligned student software engineering project.

Many students have used GitHub only as a place to store code. In this model, GitHub is treated as an engineering evidence system.

The repository contains not only code, but also requirements, planning, architecture, reviews, testing evidence, AI-use documentation, release records, and operational maturity artifacts.

### Core Idea

The GitHub repository is the project's engineering control center.

The README is the front door.

The repository should make intent, work, review, evidence, release judgment, and operational learning visible to another engineer.

### Repository-Centered Engineering Doctrine

| Principle | Meaning |
|---|---|
| Everything important leaves evidence | Important engineering claims should point to inspectable repository evidence. |
| The README is the front door | A reviewer should know where to begin and where proof lives. |
| No traceability, no merge | Meaningful changes should connect intent, work, review, tests, and evidence. |
| Merges require engineering judgment | Pull requests should explain why changes were considered safe enough to merge into the operational system. |
| AI proposes; engineers verify | AI-generated or AI-assisted artifacts must be disclosed, reviewed, and verified. |
| A demo is not operational proof | Working once is not the same as being reviewable, testable, governable, observable, or supportable. |
| Honest evidence beats polished claims | Known limitations, failed checks, deferred work, and residual risks should be visible. |

### Starter Kit Structure

The starter kit provides the Day-1 repository scaffold.

Students should not treat it as busywork.

Each folder has a purpose, and each Markdown file is a starting point for engineering evidence.

Repository structure should support real engineering work and reviewability, not create the appearance of maturity without supporting evidence.

comp330-f26-teamX-projectname/ README.md .gitignore

.github/ ISSUE_TEMPLATE/ pull_request_template.md workflows/ ci.yml

docs/ team/ requirements/ planning/ architecture/ reviews/ testing/ quality/ ai/ release/ observability/ security/ operations/

src/ tests/ test-evidence/ scripts/ data/

**Folder Purpose Quick Reference**

| Folder or Area | Purpose |
|---|---|
| `/docs/team` | Team charter, roles, working agreements, ownership, and coordination rules. |
| `/docs/requirements` | Requirements, acceptance criteria, assumptions, and open questions. |
| `/docs/planning` | Scope, task plan, estimates, schedule, risk register, and traceability. |
| `/docs/architecture` | Architecture overview, interfaces, component responsibilities, diagrams, boundaries, ADRs, and major engineering decisions. |
| `/docs/reviews` | Architecture, code, AI-code, release readiness, and operational review evidence. |
| `/docs/testing` | Test strategy, test plan, test cases, CI evidence, and validation summaries. |
| `/docs/quality` | Defect logs and defect review summaries. |
| `/docs/ai` | AI policy, AI-use log, and AI verification notes. |
| `/docs/release` | Release notes, known limitations, demo script, and release traceability. |
| `/docs/observability` | Runtime evidence, logs, metrics, and operational assumptions. |
| `/docs/security` | Security/governance checklist, data handling, and permission boundaries. |
| `/docs/operations` | Runbooks, incident response notes, postmortems, and recovery guidance. |
| `/src` | Source code. |
| `/tests` | Automated tests. |
| `/test-evidence` | Manual testing evidence, screenshots, logs, or evaluation outputs. |
| `/.github` | Issue templates, pull request template, and GitHub Actions workflows. |

**Assignment Progression**

| Assignment | Repository Areas Emphasized | What This Proves |
|---|---|---|
| A1: Repository Launch | `README.md`, `/docs/team`, `/docs/requirements`, `/docs/ai`, `/.github` | Team is formed, roles are visible, AI policy exists, and the repository is ready to operate. |

| Assignment | Repository Areas Emphasized | What This Proves |
|---|---|---|
| A2: Planning and Estimation | `/docs/planning`, `/docs/requirements`, Issues, Milestones | Scope, tasks, estimates, risks, schedule, commitments, and traceability are visible. |
| A3: Architecture and Review | `/docs/architecture`, `/docs/reviews`, `/docs/security` | Components, boundaries, interfaces, AI/tool boundaries, governance points, and risks are reviewable. |
| A4: Construction and Integration | `/src`, `/tests`, `/.github/workflows`, `/docs/testing`, `/docs/ai` | Implementation is controlled through issues, branches, PRs, reviews, CI/CD, tests, and AI-use evidence. |
| A5: Cycle 1 Release | `/docs/release`, `/docs/testing`, `/test-evidence`, `/docs/quality` | Cycle 1 release claims are supported by test evidence, defects, known limitations, release notes, and release/version evidence. |
| A6: Final Release and Maturity | `/docs/operations`, `/docs/observability`, `/docs/security`, `/docs/release` | Operational readiness, governance, observability, recovery expectations, and release maturity are visible. |

### README Expectations

The root README should include:

- Project title, team identity, course, semester, and repository URL.
- Project purpose, intended users, and supported workflow.
- Current release status: what works, what is incomplete, and known limitations.
- How to run or inspect the system, including setup and test commands.
- Evidence index linking directly to requirements, planning, architecture, testing, AI-use, release, security, observability, and operations artifacts.
- Operational expectations, observability assumptions, or recovery guidance when appropriate to the system maturity level.
- Repository workflow expectations: issues → branches → pull requests → review → validation checks → merge → evidence updated.

Repository evidence should remain reviewable without requiring verbal clarification from the original team.

### What Not to Put in the Repository

Do not store:

- Real student records, grades, private university data, credentials, secrets, API keys, passwords, or tokens.
- Private peer reviews.
- Large generated folders, dependency folders, build output, temporary files, or local environment files unless explicitly justified.
- Unrelated screenshots, old drafts, or files that cannot be traced to project evidence.
- AI-generated or AI-assisted work that the team cannot explain, test, review, and maintain.

### Repository Review Checklist

- ☐ README explains project purpose, current status, run/test instructions, and evidence locations.
- ☐ Repository structure aligns with current assignment expectations.
- ☐ Issues and pull requests show current and completed work.
- ☐ Tests or validation evidence support important claims.
- ☐ AI-use log is current and honest.
- ☐ Known limitations, defects, and risks are visible.
- ☐ No secrets, private data, peer reviews, or unsafe files are stored in the repository.
- ☐ Another engineering team could inspect, review, and operationally understand the repository without relying on private explanations.

### Final Takeaway

A professional repository is not merely a code dump.

It is the visible engineering control system for planning, review, testing, governance, release readiness, and operational learning.

## GitHub Repository Setup Guidance

**Classification:** ETIS Educational Ecosystem Asset
**Category:** Workflow
**Origin:** Loyola University Chicago COMP 330 — Software Engineering
**Authoritative Format:** Markdown

### Core Idea

Your team GitHub repository is the project control center.

It is where the team creates the authoritative evidence trail for work, review, testing, AI use, and release readiness.

This document explains how to create and connect the repository. The separate repository structure and README guidance explains what must live inside it.

### 1. Purpose of This Guidance

Each COMP 330 team must create a GitHub repository early in the semester. The repository should be created as one of the first team formation actions by Week 2, before serious project work begins.

The purpose is not simply file storage. The repository is the shared engineering workspace where lifecycle-traceable evidence is created through issues, branches, pull requests, reviews, testing, and releases.

This document focuses on repository setup. It does not define the required folder structure, README contents, or detailed artifact layout. Those expectations are covered in the separate Repository Structure and README Guidance document.

Use this document first to create the repository correctly. Then use the structure document to organize the repository professionally.

The two documents should be used together:

- This document explains how to establish and operate the repository.
- The Repository Structure and README Guidance document defines the professional engineering layout and evidence organization expected throughout the semester.

### 2. Required Repository Setup Decision

Each team must create one public GitHub repository for the semester project unless the instructor explicitly approves a different arrangement.

Public repositories make evidence easier to inspect during phase-gate reviews and reduce access problems during grading.

The repository should be created by the team lead or another agreed owner during team launch.

All team members must be added as collaborators or given appropriate access through the repository or organization.

The repository URL must be submitted in assignments and used for phase-gate reviews.

The repository must not contain:

- Real private data
- Credentials
- Secrets
- Tokens

- Student records
- Grades
- Live university information

Use synthetic data for examples, demos, tests, and screenshots.

### 3. Standard Repository Naming Convention

Use a predictable repository name so the instructor can identify teams quickly and so repositories remain professional.

Use lowercase letters, numbers, and hyphens only.

| Item | Required Convention |
|---|---|
| Recommended format | `comp330-f26-teamNN-project-name` |
| Example default project | `comp330-f26-team03-campusconnect` |
| Example alternative project | `comp330-f26-team07-soccer-player-analysis` |

If team numbers are not assigned yet, use a temporary descriptive name and rename the repository after team numbers are assigned.

Do not use joke names, ambiguous names, or personal account names as the project identity.

### 4. Create the Repository on GitHub

1. Sign in to GitHub using an account you can reliably access throughout the semester.
2. Create a new repository using the standard naming convention above.
3. Set visibility to **Public** unless the instructor approves a private repository.
4. Initialize the repository with a README if GitHub offers that option. The README can be replaced later using the repository structure guidance.
5. The repository README should become the professional front door to the project and link to major engineering evidence, release artifacts, and operational resources.
6. Add a `.gitignore` appropriate for the technology stack if the team already knows the stack. If unsure, add it later after the architecture and development plan are clearer.
7. Do not add real credentials, API keys, secrets, or private data to the repository.
8. After creation, copy the repository HTTPS clone URL ending in `.git` and record it in the team launch evidence.

### 5. Repository URL Required for Assignments and Phase-Gate Reviews

Assignments act as phase-gate reviews and require the repository URL.

The preferred URL for setup and cloning is the HTTPS clone URL ending in `.git`.

GitHub remote URLs are commonly HTTPS URLs such as:

https://github.com/user/repo.git

or SSH URLs such as:

git@github.com:user/repo.git

#### How to Find the `.git` URL

1. Open the team repository in GitHub.
2. Click the green **Code** button.

3. Select **HTTPS** unless the team is intentionally using SSH.
4. Copy the URL that ends with `.git`.
5. Paste that URL into the team launch document, Assignment 1 submission, and repository evidence index as required.

| URL Type | Example |
|---|---|
| HTTPS clone URL | `https://github.com/loyola-comp330/comp330-fall2` |
| SSH clone URL | `git@github.com:loyola-comp330/comp330-fall2026-` |

### 6. Add Team Members and Establish Access

Every team member must be able to:

- Clone the repository
- Create branches
- Push branches
- Open pull requests
- Review pull requests
- Update evidence artifacts

At least two people should have administrative or maintainer access so the team is not blocked if one person is unavailable.

Do not share GitHub passwords, personal tokens, or SSH private keys.

If a student cannot access the repository, the team should treat that as a project blocker and resolve it immediately.

### 7. Recommended GitHub Tool Ecosystem

Students may use any reasonable workflow that produces visible repository evidence.

The course does not require one specific tool interface, but teams should be familiar with the common GitHub ecosystem.

| Tool or Surface | What It Is Useful For | Course Expectation |
|---|---|---|
| GitHub website | Repository creation, issues, pull requests, reviews, Actions, settings, and evidence inspection. | Required. Students must be comfortable navigating the repository in the browser. |
| GitHub Desktop | Visual cloning, branching, committing, pulling, pushing, and pull request support for students less comfortable with command line. | Allowed and recommended for students who want a visual Git workflow. |
| Command-line Git | Direct professional workflow for clone, branch, commit, push, pull, merge, and troubleshooting. | Strongly recommended. Appendix B provides a basic workflow. |

| Tool or Surface | What It Is Useful For | Course Expectation |
|---|---|---|
| Visual Studio Code | Editing source code and documentation, integrated terminal, Git view, extensions, and GitHub Pull Requests extension. | Allowed. Do not rely on extensions the team cannot explain or maintain. |
| GitHub Issues and Projects | Task tracking, defects, enhancements, backlog, assignment, and traceability. | Expected for meaningful work tracking, ownership visibility, defect management, and engineering traceability. |
| GitHub Actions | Basic automated build/test validation supporting controlled integration, release confidence, and operational readiness. | Introduced during construction and integration; expected to support visible build/test evidence during release readiness and maturity reviews. |
| GitHub Copilot / AI tools | Code suggestions, explanations, review support, test ideas, documentation drafts, pull request exploration, and implementation assistance. | Allowed when disclosed, reviewed, verified, and owned by the team. |
| Logs, CI artifacts, and runtime evidence | Observability, debugging, runtime validation, failure investigation, and release confidence. | Introduced progressively as the system matures through Cycle 2 operational readiness. |

### 8. Optional Microsoft Teams Integration

Teams may coordinate in Microsoft Teams, Zoom, Discord, text chat, or another agreed channel, but GitHub remains the authoritative engineering record.

Microsoft Teams integration with GitHub is optional, not required.

If a team uses Microsoft Teams, GitHub integration can help surface repository activity in a Teams channel. Teams may use notifications for issues, pull requests, commits, code reviews, and Actions workflow events.

Recommended use:

- Receive notifications about new issues
- Receive notifications about pull requests
- Receive notifications about reviews
- Receive notifications about CI/CD failures

Not recommended:

- Treating Teams chat as the only place where engineering decisions live

Course rule:

> Decisions made in chat must be reflected in GitHub issues, pull requests, decision records, or other repository evidence.

### 9. GitHub and AI Integration Points

GitHub is increasingly an AI-assisted engineering environment.

Teams may use GitHub Copilot or other approved AI tools, but AI output must remain subject to human engineering judgment.

GitHub Copilot can assist with:

- Coding
- Code explanation
- Pull request understanding
- Code review workflows
- Test ideas
- Documentation drafts

Teams may use AI to draft, explain, critique, test, and review work.

Teams must not allow AI to become an unreviewed author of project truth.

Meaningful AI assistance must be disclosed in the AI-use log.

AI-generated code should be reviewed more carefully, not less carefully.

Teams remain responsible for the long-term maintainability, security, correctness, and operational consequences of AI-assisted work.

Never commit secrets, private data, or sensitive student information into AI prompts or the repository.

### 10. Minimum Setup Checklist

- ☐ Public GitHub repository created using the course naming convention.
- ☐ All team members have access and can clone the repository.
- ☐ Repository HTTPS clone URL ending in `.git` has been copied and recorded.
- ☐ Initial README exists, even if it is temporary.
- ☐ README links to major engineering evidence locations as the repository matures.
- ☐ No real private data, credentials, secrets, or live university data is stored in the repository.
- ☐ Team has agreed whether to use GitHub Desktop, command-line Git, Visual Studio Code, or a mixed workflow.
- ☐ Initial issue tracker use has begun for team setup tasks.
- ☐ Team knows that the separate Repository Structure and README Guidance document controls the required folder and artifact layout.
- ☐ Assignment 1 launch evidence points to the repository URL and authoritative evidence locations.

### 11. Common Setup Mistakes to Avoid

| Mistake | Why It Hurts the Team |
|---|---|
| Creating the repository late | The team loses early evidence and starts the project without a shared control system. |
| Making the repository private without approval | The instructor may not be able to inspect evidence during phase-gate reviews. |
| Using inconsistent or unclear naming | Repositories become harder to identify and manage. |
| Only one person has access | The team is blocked when that person is unavailable. |
| Uploading files manually without learning Git workflow | The team misses the professional issue-branch-pull request-review-traceability-evidence model used to control engineering change. |

| Mistake | Why It Hurts the Team |
|---|---|
| Putting decisions only in chat | Important engineering decisions become invisible and unreviewable. |
| Committing secrets or real data | Creates unnecessary privacy, security, and governance risk. |

### Appendix A: Git and GitHub for Students New to the Tools

Git and GitHub are related, but they are not the same thing.

| Concept | Plain-English Explanation |
|---|---|
| Git | The version control tool. It tracks changes, branches, commits, merges, and history. |
| GitHub | A cloud platform that hosts Git repositories and adds collaboration features such as issues, pull requests, reviews, Actions, permissions, and project visibility. |
| Local repository | The copy of the project on your computer. |
| Remote repository | The shared copy hosted on GitHub. The default remote is usually named `origin`. |

A simple mental model:

> Git tracks the work. GitHub hosts the shared project and makes teamwork visible.

### Appendix B: Command-Line Git Workflow for Beginners

Students may use GitHub Desktop or Visual Studio Code, but every software engineer should understand the command-line workflow.

The sequence below is intentionally basic.

**1. Confirm Git Is Installed** bash git –version

**2. Configure Your Name and Email Once** bash git config –global user.name “Your Name” git config –global user.email “your-luc-email@luc.edu”

**3. Clone the Team Repository Locally** bash git clone https://github.com/OWNER/REPOSITORY.git cd REPOSITORY

**4. Start from the Trusted Main Branch** bash git checkout main git pull origin main

**5. Create a Branch Tied to an Issue** bash git checkout -b issue-12-add-request-form

**6. Inspect Changed Files Before Committing** bash git status git diff

**7. Stage and Commit Your Work** bash git add . git commit -m “Add request form for issue #12”

**8. Push Your Branch to GitHub** bash git push -u origin issue-12-add-request-form

**9. Open a Pull Request on GitHub** Open GitHub in the browser, compare your branch to `main`, and create a pull request.

Link the issue using text such as:

Closes #12

**10. After Review and Merge, Update Local Main** bash git checkout main git pull origin main

**11. Delete the Local Branch After Merge** bash git branch -d issue-12-add-request-form

### Appendix C: Pull Request Basics

A pull request is the review gate between individual work and team-owned work.

Before a change enters `main`, the team should be able to answer:

- What changed?
- Why did it change?
- Who reviewed it?
- What evidence proves it works?
- Where is that evidence stored?

Engineering evidence associated with pull requests should be reviewable without requiring verbal clarification from the original author.

Open a pull request for meaningful code, documentation, testing, configuration, or evidence changes.

Link the pull request to the issue or task that explains why the change exists.

Keep pull requests small enough to review honestly.

Include test evidence, screenshots, or documentation links when relevant.

Disclose AI assistance when AI helped produce code, tests, documents, diagrams, or review notes.

Do not merge your own work without review unless the instructor explicitly permits an exception.

Pull requests should explain not only what changed, but also why the change was safe enough to merge.

### Appendix D: How This Setup Connects to Course Evidence

The repository setup supports the course traceability model.

The core rule is:

No issue → no branch → no pull request → no review → no merge.

The workflow exists to create visible engineering traceability showing:

- Why work was performed
- How it was reviewed
- What changed
- What risks were identified
- Why the change was accepted

Issues capture tasks, defects, enhancements, and questions.

Branches isolate unfinished work.

Pull requests create the review gate.

Reviews challenge correctness, scope, security, maintainability, AI use, and evidence.

Merges move approved work into the shared version of truth.

Releases and repository tags identify the specific repository state associated with major demonstrations, presentations, and release reviews.

Documentation and test evidence show that the team can defend the change later.

### Appendix E: Useful Official References

These official GitHub resources may help students who want more detail beyond this course setup guidance.

- GitHub Docs: Cloning a repository
- GitHub Docs: About remote repositories and remote URLs
- GitHub Docs: GitHub Desktop documentation
- GitHub Docs: Integrating GitHub with Microsoft Teams
- GitHub Docs: GitHub Copilot documentation and Copilot code review

### Final Takeaway

Create the repository early.

Make it public unless instructed otherwise.

Name it consistently.

Add the whole team.

Copy the `.git` clone URL.

Treat GitHub as the engineering control system for the semester.

The repository is where the team proves, reviews, governs, and operationally defends its engineering work.

## Pull Request Expectations Mini Guide

**Classification:** ETIS Educational Ecosystem Asset
**Category:** Workflow
**Origin:** Loyola University Chicago COMP 330 — Software Engineering
**Authoritative Format:** Markdown
**Typical Repository Location:** `/docs/workflow/pull-request-expectations.md`

### Purpose

Use this guide when opening, reviewing, and merging pull requests in a team repository.

Pull requests are where private work becomes reviewable team-owned engineering evidence.

A pull request should explain what changed, why it changed, how it was tested, what risks remain, and what AI assistance was used or verified.

Important pull requests should remain traceable through linked requirements, issues, tests, defects, ADRs, release notes, AI-use evidence, operational artifacts, and known limitations when appropriate.

Another engineering team should be able to understand what changed, why it changed, what evidence supports it, what risks remain, and how the change affects the release without relying on private explanation from the original author.

### When to Open a Pull Request

Open a pull request when a focused unit of work is ready for team review.

Teams should:

- Keep pull requests small enough for another student to review seriously.
- Link the pull request to a requirement, issue, task, defect, or release item whenever possible.
- Avoid using pull requests as a last-minute dumping ground for unrelated changes.
- Prefer smaller, reviewable pull requests over large multi-purpose merges that reviewers cannot realistically evaluate.

### Pull Request Minimum Content

| Required Item | What Reviewers Should See |
|---|---|
| Purpose | A short explanation of the problem, requirement, defect, or task this pull request addresses. |
| Linked evidence | Issue, requirement, task, defect, ADR, test case, or release note connection when applicable. |
| Change summary | A concise list of the files or behaviors changed. |
| Testing / verification | Tests run, manual checks performed, CI/CD results, screenshots, logs, or known test gaps. |
| Risk notes | Anything reviewers should inspect carefully: architecture fit, edge cases, security, data handling, AI output, dependency changes, operational impact, observability impact, rollback concerns, supportability implications, or known limitations. |
| AI disclosure | What AI helped produce or review, what the team accepted or changed, and how the result was verified. |

### Good Pull Request Size

| Strong Pattern | Weak Pattern |
|---|---|
| One focused issue or feature path. | Several unrelated features, fixes, formatting changes, and documentation edits mixed together. |
| Reviewable diff that another team member can understand in one sitting. | Large diff that reviewers approve without actually understanding. |
| Clear test or evidence connection. | No clear way to tell whether the change was validated. |
| Architecture or requirement impact is explained. | Design impact is hidden inside code changes. |
| Review discussion reveals real engineering questions, clarification, or validation. | "LGTM" approval without meaningful review evidence. |

### Author Responsibilities Before Requesting Review

Before requesting review, the author should:

- Confirm the branch builds or runs as expected.
- Run relevant tests or clearly explain why tests were not run.
- Update related documentation, requirements, test evidence, defect logs, AI-use logs, or release notes when needed.
- Identify known limitations, risks, or follow-up work honestly.
- Ask reviewers to focus on specific concerns when the change has risk.
- Ensure the branch does not introduce undocumented operational assumptions, hidden configuration dependencies, or unrecoverable deployment behavior.

### Reviewer Responsibilities

| Review Area | Reviewer Question |
|---|---|
| Requirement fit | Does the change match the intended requirement, acceptance criterion, defect, or task? |
| Architecture fit | Does the change respect component responsibilities, interfaces, data ownership, and design decisions? |
| Correctness | Does the change handle normal, boundary, and failure cases? |
| Testing evidence | Do the tests or manual evidence support the claim? What remains untested? |
| Security / governance | Does the change affect permissions, data handling, approval boundaries, dependencies, or AI/tool authority? |
| Maintainability | Can the team understand, maintain, diagnose, and safely change this later? |
| Operational impact | Are observability, rollback, recovery, supportability, or deployment concerns visible when relevant? |
| AI accountability | If AI helped, was the output reviewed, modified where needed, tested, and owned by the team? |

### Merge Expectations

Do not merge only because the demo path works once.

Do not merge unresolved high-risk review concerns.

Do not merge AI-generated code that the team cannot explain, test, or maintain.

Merge only when the change is reviewed, validated, traceable, and safe enough to keep.

If a known risk remains, document it in the appropriate artifact before or immediately after merge.

Do not merge changes that reviewers cannot adequately explain, test, diagnose, or support operationally.

### Recommended Pull Request Template

The sample text below may be copied into `.github/pull_request_template.md`.

markdown ### Purpose

What issue, requirement, defect, or task does this pull request address?

### Summary of Changes

-
-
-

### Evidence / Testing

- Tests run:
- Manual checks:
- CI/CD evidence:
- Known gaps:
- Operational or observability impact:

### Review Focus

Please pay special attention to:

-

### AI Assistance

- AI used? Yes / No
- What AI helped with:
- What humans changed or rejected:
- Verification performed:

### Risks / Follow-Up

- Known limitations:
- Deferred work:
- Related release note or defect entry:

### Quick Quality Gate

| Before Merge | Yes / No / N/A |
| --- | --- |
| Purpose and scope are clear. | |
| Linked issue, requirement, defect, or task is included where applicable. | |
| Relevant tests or manual evidence are documented. | |
| CI/CD status is visible or limitations are explained. | |
| AI assistance is disclosed and verified where applicable. | |
| Security, dependency, data, or governance concerns are reviewed. | |
| Operational, observability, rollback, or deployment concerns are documented where relevant. | |
| Documentation, test evidence, or release notes are updated if affected. | |
| Remaining risks or deferred work are visible. | |

**Final Takeaway**

A pull request is not just a code diff.

It is a professional review gate that connects intent, implementation, evidence, AI accountability, and merge judgment.

## Repository Starter Kit Setup Guide

**Classification:** ETIS Educational Ecosystem Asset
**Category:** Workflow
**Origin:** Loyola University Chicago COMP 330 — Software Engineering
**Authoritative Format:** Markdown
**Typical Repository Location:** Course starter kit documentation

### Purpose

This guide provides step-by-step setup instructions for using an ETIS-aligned student repository starter kit professionally.

The team repository should be created once, organized correctly, and used as the authoritative engineering evidence system for the project.

Do not treat GitHub as file storage only.

For broader GitHub workflow, CI/CD, pull request, issue, and AI-assisted development guidance, also use the GitHub repository setup guidance and repository structure guidance.

### Before You Begin

- One student should serve as initial repository creator, but the repository belongs to the team.
- Use the official course starter kit file provided by the instructor.
- Use a clear repository name, such as `comp330-f26-teamX-projectname`.
- Create a public repository unless instructed otherwise.
- Do not upload private data, secrets, API keys, tokens, or peer reviews.

### Recommended Professional Setup Path

This is the cleanest Day-1 workflow for most teams.

1. Download the official starter kit archive from the course site.
2. Create a new empty GitHub repository using the team naming convention.
3. Clone the empty repository to your computer.
4. Extract the starter kit into the cloned repository folder.
5. Review `README.md` and update the project name, team number, members, and repository URL.
6. After the starter structure is committed, use the repository structure and README guidance to keep the README, `/docs` folders, templates, and evidence locations current throughout the semester.
7. Commit the starter structure with a meaningful message.
8. Push the initial commit to GitHub.
9. Invite all team members as collaborators.
10. Confirm every team member can clone, pull, create a branch, and open a pull request.

### Command-Line Setup

bash ## 1. Clone the empty team repository git clone https://github.com//comp330-f26-teamX-projectname.git cd comp330-f26-teamX-projectname

## 2. Extract starter kit outside or inside a temporary folder

tar -xzvf ~/Downloads/comp330-f26-starter-kit.tar.gz

## 3. Copy starter contents into the repository root if needed

cp -R comp330-f26-starter-kit/. .

## 4. Confirm structure

ls -la

## 5. Stage, commit, and push

git add . git commit -m “Initialize engineering repository starter structure” git push origin main

### Alternative Setup Path: Upload Through GitHub Web UI

This path is acceptable for students who are still learning Git, but the team should move to branch/PR workflow quickly.

1. Create a new GitHub repository using the naming convention.
2. Extract the starter kit locally.
3. Use GitHub web upload to upload the starter kit files and folders.
4. Commit the uploaded structure with a clear message.
5. Clone the repository locally afterward so team members can work through Git normally.
6. Begin all meaningful work through issues, branches, pull requests, reviews, and checks.

### Repository Settings Recommended for Teams

| Setting | Recommendation | Reason |
|---|---|---|
| Visibility | Public unless instructed otherwise | Allows instructor review and avoids access friction. |
| Default branch | `main` | Standard branch name and clean workflow target. |
| Issues | Enabled | Issues are the operational work queue. |
| Actions | Enabled | Actions provide CI/CD, validation, and release-readiness evidence. |
| Projects | Optional | Useful for teams that want visual task tracking. |
| Wiki | Avoid as primary evidence store | Course evidence should live in versioned `/docs` files. |
| Branch protection | Recommended once team workflow stabilizes | Protects `main` from unreviewed changes. |

### Professional Workflow After Setup

- Create an issue for meaningful work.
- Create a branch tied to the issue, requirement, task, or evidence purpose.
- Commit changes with meaningful messages.
- Open a pull request before merging into `main`.
- Use the pull request template to document purpose, tests, risks, and AI use.
- Have another team member review meaningful changes.
- Resolve CI/check failures before merge or clearly explain why they do not apply.

- Update evidence files when changes affect requirements, architecture, testing, release notes, or AI-use records.
- Use release tags, versions, or equivalent release markers when preparing major presentation or release evidence.

### Common Student Mistakes to Avoid

- Uploading all files through the web UI every time instead of learning branch/PR workflow.
- Editing directly on `main` for meaningful changes.
- Treating `README.md` as a placeholder instead of the repository front door.
- Leaving starter files blank or stale after the project matures.
- Putting peer reviews or private data in the public repository.
- Committing AI-generated code or documentation that the team cannot explain and verify.
- Waiting until the deadline to create evidence instead of maintaining it continuously.
- Creating empty folders or stale template files only to look complete instead of maintaining real engineering evidence.

### Day-1 Success Test

By the end of setup, every team member should be able to:

- Open the repository.
- Understand the project purpose.
- Find the main evidence folders.
- Create an issue.
- Create a branch.
- Make a small change.
- Open a pull request.
- Explain the workflow.

The team should also be able to explain why the repository structure supports evidence, review, traceability, AI governance, and release readiness.

### Final Takeaway

The starter kit is not a file dump.

It is the first engineering environment the team will use to create visible, reviewable, traceable, AI-accountable, and release-defensible work.

## Engineering Workflow and Traceability Cheatsheet

**Classification:** ETIS Educational Ecosystem Asset
**Category:** Workflow
**Origin:** Loyola University Chicago COMP 330 — Software Engineering
**Authoritative Format:** Markdown
**Typical Repository Location:** `/docs/workflow/engineering-workflow-traceability.md`

### Core Idea

Professional software engineering creates explainable history.

Every meaningful change should have a visible reason, a controlled branch, a reviewed pull request, supporting evidence, and a traceable release story.

### Why Traceability Matters

Traceability is the connection between intent, work, review, evidence, and release.

It lets another engineer answer:

- What changed?
- Why did it change?
- Who reviewed it?
- What evidence proves it works?
- Where is that evidence stored?

Traceability prevents invisible work from entering the project.

It connects requirements to issues, pull requests, tests, reviews, release notes, and operational evidence.

It makes AI-assisted work reviewable, explainable, and accountable.

It turns a demo claim into an engineering claim.

### Simple Professional Workflow Rule

No issue → no branch → no pull request → no review → no merge.

This is not GitHub bureaucracy.

It is the minimum workflow needed to make team software visible, reviewable, and governed.

Direct commits to `main`, undocumented changes, or unreviewed AI-generated code weaken traceability and team accountability.

### Traceability Chain

| Step | Artifact | Purpose |
|---|---|---|
| 1 | Requirement | What the system must do and how success is judged. |
| 2 | Issue | The specific task, defect, enhancement, or question to work on. |
| 3 | Branch | Isolated workspace tied to the issue. |

| | Step | Artifact | Purpose |
|---|---|---|---|
| | 4 | Pull Request | Reviewable proposed change linked to issue and evidence. |
| | 5 | Review + Tests | Human review, CI status, test notes, risk comments, AI disclosure. |
| | 6 | Release Evidence | Release notes, known limitations, demo script, postmortem learning. |

**Example End-to-End Flow**

| Artifact | Example |
|---|---|
| Requirement | REQ-03: Reviewer can approve or reject a request only after required fields are complete. |
| GitHub Issue | #42: Enforce required-field validation before approval. |
| Branch | `issue-42-approval-validation` |
| Pull Request | PR #18: Adds approval validation and tests. Includes `Closes #42`. |
| Review Evidence | Reviewer checks requirement fit, code, tests, security, AI use, and architecture consistency. |
| Test Evidence | T-07: Missing required fields block approval; CI run passes. |
| Release Evidence | `release-notes.md` identifies the approved workflow, validation evidence, and known limitations. |

**What Goes Where in GitHub**

| Item | Where It Lives | Purpose |
|---|---|---|
| Ideas, tasks, defects, enhancements | GitHub Issues | Operational work queue. Issues explain why work exists. |
| Branches | Git branches | Safe isolated work areas tied to issues. |
| Pull requests | GitHub Pull Requests tab | Review gate before changes enter main. |
| Requirements | `/docs/requirements/` | Defines what the system must do and how acceptance is judged. |
| Planning and traceability | `/docs/planning/` | Scope, task plan, estimates, risks, schedule, traceability, and re-estimation. |
| Architecture and decisions | `/docs/architecture/` and `/docs/decisions/` | System structure and decision reasoning. |

| Item | Where It Lives | Purpose |
|---|---|---|
| Testing and quality | `/docs/testing/` and `/docs/quality/` | Test strategy, test evidence, defects, CI evidence, and validation notes. |
| AI accountability | `/docs/ai/` | AI policy, AI-use log, verification notes, and disclosure evidence. |
| Release evidence | `/docs/release/` | Release notes, known limitations, presentation index, demo script, and final evidence package. |

### Issue, Branch, and Pull Request Naming

| Artifact | Recommended Form | Example |
|---|---|---|
| Issue title | Short action-oriented title | Fix approval validation edge case |
| Branch name | `issue-[number]-short-description` | `issue-42-approval-validation` |
| Pull request title | Clear summary of the change | Fix approval validation for required fields |
| Commit message | Concise change statement with issue reference | Fix approval validation for issue #42 |

### Pull Request Evidence Checklist

A pull request should make the change reviewable without requiring a private explanation in chat.

- Linked issue number, such as `Closes #42`.
- Short summary of what changed and why.
- Requirement or artifact path affected by the change.
- Tests run, CI status, or manual validation evidence.
- AI assistance disclosure, if applicable.
- Known limitations, risks, or follow-up work.
- Reviewer comments and author responses when changes are requested.

### Review Expectations

| Reviewer Asks | Why It Matters |
|---|---|
| Does this satisfy the issue or requirement? | Prevents random work from entering the system. |
| Is the change small enough to understand? | Large changes hide defects and weaken accountability. |
| Are tests or evidence included? | A claim without evidence is just an opinion. |
| Does the change fit the architecture? | Prevents design drift and accidental complexity. |
| Is AI use disclosed and verified? | Generated work still belongs to the team. |
| Can the team explain and maintain this later? | Professional engineering must survive beyond the author of the change. |

### Common Mistakes to Avoid

| Mistake | Why It Hurts the Team |
|---|---|
| Working directly on main | No safe separation of unfinished work and no review gate. |
| Doing work only discussed in chat | The work becomes invisible to the repository evidence trail. |
| Opening a pull request with no issue link | Reviewers cannot trace why the change exists. |
| Merging without test or validation evidence | The team relies on confidence instead of proof. |
| Using AI-generated code no one can explain | The team owns the risk even if AI produced the text. |
| Creating many empty placeholder files | This creates artifact theater instead of useful engineering evidence. |

**Final Student Rule**

Before merge, answer five questions:

What changed? Why did it change? Who reviewed it? What evidence proves it works? Where is that evidence stored?

**Final Takeaway**

The repository is not just file storage.

It is the team's engineering evidence system.

Strong teams make their work easy to inspect, review, explain, and improve.

## Definition of Done Checklist

**Classification:** ETIS Educational Ecosystem Asset
**Category:** Workflow
**Origin:** Loyola University Chicago COMP 330 — Software Engineering
**Authoritative Format:** Markdown
**Typical Repository Location:** `/docs/release/definition-of-done.md`

### Purpose

Use this checklist before each major cycle delivery, release package, or presentation.

The checklist helps a team decide whether work is actually done, not merely coded, demonstrated once, or documented after the fact.

The repository remains the authoritative evidence source.

Another engineering team should be able to inspect the repository evidence and understand what changed, what was reviewed, what was tested, what risks remain, and why the current release is defensible.

### Team and Delivery Information

| Field | Team Response |
|---|---|
| Team Number / Name | |
| Repository URL | |
| Delivery / Assignment | |
| Release / Cycle | |
| Checklist Date | |
| Checklist Owner | |

### How to Use This Checklist

- Complete this checklist before submitting a cycle package or presenting a release.
- Mark an item done only when the team can point to repository evidence.
- If an item is not applicable, mark it `N/A` and briefly explain why.
- If an item is incomplete, record the gap, owner, risk, and next action.
- Do not treat the checklist as paperwork. Use it as a release-readiness control.
- A completed checklist should reflect the actual engineering state of the release, including known limitations, unresolved risks, operational concerns, and deferred work.

### Definition of Done Checklist

A delivery is done when the team can explain what changed, why it changed, what evidence supports it, what risks remain, and who owns the next action.

| Done? | Check Area | Definition of Done | Evidence Location | Owner |
|---|---|---|---|---|
| [ ] | Scope and Requirements | Completed work is tied to current requirements, acceptance criteria, and intentionally deferred scope. | `/docs/requirements/`; `/docs/planning/traceability.md` | |
| [ ] | Task and Issue Traceability | Major work items connect to issues/tasks, owners, branches, commits, and pull requests. | GitHub Issues, PRs, `/docs/planning/traceability.md` | |
| [ ] | Architecture Fit | Implementation remains consistent with the architecture or documented architecture changes and ADRs. | `/docs/architecture/`; `/docs/decisions/` | |
| [ ] | Code and Configuration | Code/config changes are committed, organized, readable, and free of obvious dead code, secrets, or local-only assumptions. | `/src/`; `/scripts/`; config files; PR diffs | |
| [ ] | Pull Request Review | Meaningful changes were reviewed before merge, with comments, risks, decisions, or rationale visible. | GitHub Pull Requests; review comments | |
| [ ] | Testing Evidence | Relevant unit, integration, system, manual, regression, or AI validation evidence exists and is current. | `/docs/testing/`; `/tests/`; `/test-evidence/` | |

| Done? | Check Area | Definition of Done | Evidence Location | Owner |
|---|---|---|---|---|
| [ ] | CI/CD Evidence | Build/test workflow status is visible; failed, skipped, or missing checks are explained. | `/.github/workflows/`; `/docs/testing/ci-evidence.md` | |
| [ ] | Defect Status | Known defects are logged, classified, owned, and dispositioned as fixed, mitigated, deferred, accepted, or removed from scope. | `/docs/quality/defect-log.md` | |
| [ ] | AI Use and Verification | AI-assisted work is disclosed, reviewed, modified as needed, and verified by humans. | `/docs/ai/ai-use-log.md`; PR notes | |
| [ ] | Security and Data Handling | Sensitive data, permissions, dependencies, external services, and AI/tool boundaries have been reviewed. | `/docs/security/`; risk register | |
| [ ] | Observability / Operations | Important runtime behavior, failures, setup, recovery, and operational support notes are documented where relevant. | `/docs/observability/`; `/docs/operations/runbook.md` | |

| Done? | Check Area | Definition of Done | Evidence Location | Owner |
|---|---|---|---|---|
| [ ] | Recovery / Rollback Readiness | Important failures, risky changes, or degraded dependencies have a documented recovery, rollback, disablement, or safe-mode path where appropriate. | `/docs/operations/runbook.md`; release notes; rollback notes | |
| [ ] | Release Notes / Known Limits | Release notes explain what changed, what works, known limitations, residual risks, and next steps. | `/docs/release/release-notes.md`; known limitations | |
| [ ] | README / Navigation | README and repository navigation help another engineer understand, run, test, and review the project. | `README.md`; docs README files | |
| [ ] | Presentation Readiness | Presentation claims are backed by repository evidence, demo path is rehearsed, and backup evidence exists. | `/docs/release/presentation-index.md`; demo script | |

**Gap / Risk Log**

Use this table for any checklist item that is incomplete, risky, deferred, or intentionally accepted for the current delivery.

| Item | Gap / Risk | Impact | Owner | Disposition | Next Action / Evidence |
|---|---|---|---|---|---|

### Final Readiness Decision

Select one.

- ☐ Ready to submit / present: all critical items are complete, evidence exists, and remaining risks are documented.
- ☐ Ready with known limitations: the release is defensible, but limitations, mitigations, and deferred work must be clearly stated.
- ☐ Not ready: a critical build, test, demo, security, evidence, or ownership gap must be corrected before submission or presentation.

| Field | Team Response |
|---|---|
| Final Readiness Decision | |
| Decision Rationale | |
| Approved By / Team Role | |
| Date | |

The rationale should explain remaining risks, mitigations, operational concerns, deferred work, and why the current release remains defensible.

### Final Takeaway

Done means reviewable, testable, traceable, governable, and explainable.

If the team cannot point to evidence, the work is not professionally done yet.

# Appendix

## COMP330 Starter Kit Reference

## COMP 330 Engineering Starter Kit

Repository-centered engineering environment for ETIS-based software engineering projects.

### Overview

This repository serves as the official starter kit for **COMP 330 — Software Engineering** at Loyola University Chicago.

This repository is the flagship ETIS implementation.

It demonstrates how ETIS educational systems can be operationalized within a real software engineering course.

The flagship implementation proves the doctrine.

It does not become the doctrine.

Future institutions should inherit ETIS principles while adapting implementations to their own educational environments.

It provides a structured engineering environment that students use to initialize their team project at the beginning of **Cycle 1**.

The starter kit is built upon the principles of **Engineering Trustworthy Intelligent Systems (ETIS)** and is designed to establish professional engineering practices from the first day of the project.

Students are expected to use this repository as the foundation for all project work throughout the semester.

The repository is intentionally designed to support engineering evidence, reviewability, traceability, AI accountability, release readiness, and operational thinking.

### Educational Purpose

This repository exists to help students experience software engineering as a professional engineering discipline rather than a coding exercise.

The goal is not simply to build a working application.

The goal is to build a system that can be:

- understood,
- reviewed,
- validated,
- governed,
- operated,
- improved,
- and defended.

The repository serves as the authoritative engineering record for the project.

This repository also supports the ETIS professional transformation pathway.

Student

↓

Responsible Engineer

↓

Reviewer

↓

Architect

↓

Release Defender

↓

Operational Thinker

↓

Future Trustworthy Engineer

Students should gradually stop acting like students and progressively begin acting like engineers.

### ETIS Principles

This starter kit is built upon several ETIS principles.

AI proposes; engineers verify.

Everything important leaves evidence.

Governance is architecture.

Context is control.

The model is not the system.

A demo is not operational proof.

Students are expected to apply these principles throughout the semester.

### Repository-Centered Engineering

This repository is intentionally organized to preserve engineering evidence.

The repository should contain more than source code.

It should tell the story of how the system evolved over time.

Engineering evidence may include:

- Requirements
- Assumptions
- Risks
- Architecture decisions
- AI-use records
- Reviews
- Testing evidence
- Release evidence
- Operational assumptions
- Observability notes
- Runbooks

A reviewer should be able to inspect the repository and understand:

- what was intended,
- what was built,
- what changed,

- what was tested,
- what remains risky,
- what AI assisted with,
- and why the team believes the release is defensible.

### Engineering Lifecycle

Typical engineering progression includes:

Intent
↓
Work Planned
↓
Implementation
↓
Review
↓
Verification
↓
Evidence Updated
↓
Release Readiness
↓
Operational Learning

Students are taught that engineering artifacts should remain connected throughout the project.

Traceability is not documentation overhead.

Traceability is a mechanism for understanding why work exists, how it evolves, and how engineering decisions can be defended.

Engineering work should remain visible throughout the repository.

### Two-Cycle Engineering Model

The project follows two engineering cycles.

**Cycle 1 — Can It Work?** Establish engineering foundations and build a controlled vertical slice.

Focus areas include:

- repository setup,
- team organization,
- requirements,
- planning,
- architecture,
- implementation,

- testing,
- and release readiness.

**Cycle 2 — Can It Survive?** Improve engineering maturity using evidence from Cycle 1.

Focus areas include:

- postmortems,
- stabilization,
- observability,
- governance,
- security,
- operational readiness,
- risk reduction,
- and engineering defense.

The distinction is intentional.

> A system that works once is not necessarily trustworthy.

### Repository Structure

This starter kit provides a recommended engineering structure.

.github/ docs/ src/ tests/ scripts/ data/

The structure may evolve as the project matures.

Teams may add directories when justified by project needs.

Teams should avoid unnecessary complexity or creating artifacts that are never maintained.

### Team Expectations

Teams are expected to:

- work collaboratively,
- communicate professionally,
- keep engineering evidence current,
- review each other's work,
- disclose meaningful AI use,
- connect claims to evidence,
- document important decisions,
- communicate limitations honestly,
- and continuously improve the repository.

Strong engineering teams prioritize clarity, reviewability, and accountability over feature volume.

The governing principle should be:

> Scale complexity, not accountability.

Projects may become more sophisticated over time.

Engineering accountability should remain visible from the first day of the project.

### AI Use Expectations

AI may assist with:

- brainstorming,
- requirements analysis,
- architecture critique,
- coding,
- debugging,
- testing,
- documentation,
- and summarization.

However, students remain responsible for:

- correctness,
- security,
- maintainability,
- architecture fit,
- validation,
- documentation,
- and final engineering judgment.

AI-assisted work is not accepted engineering work until humans review, verify, and own it.

### Core Rule

The repository is not a storage location.

It is an engineering system.

Every important engineering decision should leave evidence.

If a reviewer cannot understand why something exists, how it was validated, or who is accountable for it, additional evidence is likely needed.

### Relationship to ETIS

This starter kit is part of the **ETIS Educational Ecosystem**.

ETIS Framework

↓

Educational Ecosystem

↓

COMP 330 Flagship Implementation

↓

Engineering Starter Kit

↓

Student Engineering Practice

The repository exists to help students build the habits and judgment required to become trustworthy engineers.

### Final Thought

Future engineers will not be judged by how much AI-assisted output they can generate.

They will be judged by whether they can responsibly define intent, engineer context, bound authority, verify behavior, operate reality, explain decisions, and own outcomes.

This repository is where students begin practicing those responsibilities.

## AI Policy

### Purpose

This document defines how the team will responsibly use AI throughout the project.

AI should assist engineering work.

AI should not replace engineering responsibility.

The policy should remain lightweight and practical.

### Team AI Principles

We will:

- disclose meaningful AI usage
- verify AI outputs
- maintain human ownership
- identify AI risks
- avoid blind trust

### Acceptable AI Activities

Examples include:

- brainstorming
- drafting documentation
- explaining concepts
- generating examples
- proposing implementation approaches

### Higher-Risk Activities

These activities require additional verification.

Examples:

- generated code
- security recommendations
- architectural recommendations
- testing recommendations
- release recommendations

### Human Responsibilities

Humans remain responsible for:

- decisions
- verification
- risks
- outcomes

### Review Questions

- Are AI boundaries clear?
- Are responsibilities clear?
- Are higher-risk activities receiving additional review?

**Engineering Standard**

AI assistance does not transfer accountability.

## AI Use Log

### Purpose

This document records meaningful AI usage throughout the project.

The goal is visibility.

Do not document every prompt.

Document meaningful engineering assistance.

### AI Usage Entries

| Date | Work Area | AI Tool | Purpose | Human Verification Performed | Owner |
|---|---|---|---|---|---|
| | | | | | |

### What To Log

Examples:

- generated code
- architectural suggestions
- testing assistance
- documentation assistance
- debugging assistance

Do not log trivial usage.

### Review Questions

- Is AI usage visible?
- Are higher-risk uses being logged?
- Is ownership remaining human?

### Engineering Standard

Undisclosed AI dependency creates engineering risk.

## AI Verification Notes

### Purpose

This document records how AI outputs were verified.

Verification creates trust.

AI outputs should never be accepted automatically.

Keep verification lightweight and practical.

### Verification Entries

| Date | AI Output | Verification Method | Risks Found | Final Decision | Owner |
|---|---|---|---|---|---|
| | | | | | |

### Verification Methods

Examples:

- manual review
- testing
- comparison against requirements
- architecture review
- peer review

### Review Questions

- Was verification performed?
- Were risks identified?
- Is the final decision defensible?

### Engineering Standard

AI-generated work is not accepted engineering work until humans verify and own it.

## Architecture

### Purpose

This document describes how the system is intentionally organized.

Architecture should create clarity.

Architecture should simplify complexity.

Architecture should explain how major pieces of the system work together.

Keep architecture lightweight.

Do not over-engineer it.

### System Summary

Provide a brief overview.

Example:

> This system allows students to submit support requests while staff users review and manage those requests.

### Architecture Diagram

Insert or link a high-level diagram.

Examples:

- component diagram
- layered architecture
- deployment diagram

The diagram should be simple enough that another engineering team can understand it quickly.

### Major Components

| Component | Purpose |
| --- | --- |
| | |

### System Boundaries

Document important boundaries.

Examples:

- user interface
- application logic
- data storage
- external services

### Architecture Assumptions

Document assumptions.

Examples:

- synthetic data only
- no production deployment
- no external integrations

### Review Questions

- Is the architecture understandable?
- Is complexity appropriate?
- Are boundaries visible?
- Can another team quickly understand the system?

### Engineering Standard

Strong architectures make complexity understandable.

## Scope Example

**Purpose:** Defines what the team is building now, what is intentionally out of scope, and what may be deferred for later.

**Typical repository location:** `/docs/planning/scope.md`

### Engineering Guidance

Scope decisions should support engineering focus, release defensibility, operational stability, and realistic delivery expectations.

Strong engineering teams intentionally defer risky, poorly understood, weakly validated, or weakly governed features rather than overcommit beyond their evidence and review capacity.

Scope artifacts should support release decisions and engineering coordination — not justify unrealistic feature volume or presentation-driven commitments.

### Cycle Target

| Item | Description |
|---|---|
| Current cycle | Cycle 1 |
| Release goal | Demonstrate a narrow, reviewable student-support request workflow using synthetic data |
| Primary user workflow | Student submits a support request; staff user views and updates request status |
| Scope owner | Example: Planning Lead |
| Last updated | YYYY-MM-DD |

### In Scope

| ID | Scope Item | Reason Included | Related Requirement / Issue | Operational / Governance Concern |
|---|---|---|---|---|
| S-001 | Student can submit a support request | Required for the main vertical slice | FR-001 / #1 | Input validation and basic audit visibility required |
| S-002 | Staff user can view submitted requests | Needed to close the basic workflow loop | FR-002 / #2 | Access assumptions must be documented |
| S-003 | Request status can be updated manually | Supports visible lifecycle progress without over-automating | FR-003 / #3 | Status changes should be traceable |

### Out of Scope

| ID | Out-of-Scope Item | Reason Excluded | Possible Future Action |
|---|---|---|---|
| OOS-001 | Live integration with university systems | Requires private data, institutional approval, and stronger governance | Simulate with synthetic data |
| OOS-002 | Automated assignment of requests to real staff | Requires access control, workflow policy, and operational approval | Revisit after manual workflow is stable |
| OOS-003 | Production deployment | Course project does not have institutional production authorization | Demonstrate release-readiness evidence only |

### Deferred Candidates

Deferred work should often include features requiring additional testing, operational evidence, governance review, AI-boundary clarification, runtime visibility, or architectural maturity before acceptance.

| ID | Candidate | Evidence Needed Before Accepting | Target Cycle / Status |
|---|---|---|---|
| D-001 | AI category suggestion | Need Cycle 1 workflow evidence, validation plan, human approval boundary, and AI-use disclosure | Cycle 2 candidate |
| D-002 | Admin dashboard | Need stable request model and clear decision about useful metrics | Cycle 2 candidate |
| D-003 | Email notification simulation | Need test evidence and failure-handling assumptions | Backlog |

### Scope Review Questions

| Question | Current Answer |
|---|---|
| Is the scope small enough to engineer well? | Example: Yes, limited to one vertical slice |
| What is intentionally not being built? | Live integrations, production deployment, automatic routing |
| What evidence would justify expanding scope? | Passing tests, reviewed PRs, stable requirements, risk mitigation, operational visibility |
| What scope item creates the most release risk? | AI category suggestion, because validation and governance are immature |

### Reviewability Check

Another engineering team should be able to inspect this scope evidence and understand what the team chose to build, what it intentionally deferred, and why those boundaries support release defensibility.

## Risk Register Example

**Purpose:** Tracks risks that could affect scope, schedule, quality, security, governance, operational maturity, or release readiness.

**Typical repository location:** `/docs/planning/risk-register.md`

### Engineering Guidance

Risks should remain visible, actionable, and connected to engineering evidence, release decisions, workflow maturity, and operational understanding.

Significant risks should remain traceable through issues, pull requests, ADRs, release notes, postmortems, or operational evidence when appropriate.

Some risks may justify scope reduction, release delay, additional review, or operational mitigation before release approval.

### Risk Register

| Risk ID | Risk | Category | Probability | Impact | Trigger | Mitigation | Owner | Status |
|---|---|---|---|---|---|---|---|---|
| R-001 | Team underestimates integration work | Schedule / Technical | Medium | High | API and UI do not connect by checkpoint | Build early integration slice | Example: Development Lead | Open |
| R-002 | AI-generated code is accepted without understanding | AI / Quality | Medium | High | PR includes code no one can explain | Require review, validation, and AI-use log | Example: Review Lead | Open |
| R-003 | AI category suggestion lacks governance boundary | AI / Governance | Medium | High | Feature is added without human approval workflow | Defer until validation and approval boundary are documented | Example: Architecture Lead | Open |
| R-004 | Runtime failures are difficult to diagnose | Operational / Observability | Medium | Medium | Errors cannot be reproduced or explained | Improve logging and operational evidence | Example: QA Lead | Open |

| Risk ID | Risk | Category | Probability | Impact | Trigger | Mitigation | Owner | Status |
|---|---|---|---|---|---|---|---|---|
| R-005 | Scope expands beyond review capacity | Planning / Release | Medium | High | New features added without estimates or evidence plan | Require scope review and re-estimation | Example: Planning Lead | Open |

### Risk Review Notes

| Date | What Changed? | New Action Needed? | Owner |
|---|---|---|---|
| YYYY-MM-DD | AI category suggestion moved to deferred candidates | Update scope and AI-use expectations | Example: Architecture Lead |
| YYYY-MM-DD | Basic logging gap found during testing | Add operational evidence task | Example: QA Lead |

### Release-Readiness Impact

| Risk ID | Release Impact | Release Decision / Mitigation |
|---|---|---|
| R-002 | AI-assisted behavior cannot be defended if no one understands it | Require human explanation and validation before merge |
| R-004 | Failures may be hard to diagnose during demo or review | Add logs and known limitation note before release |
| R-005 | Team may present unsupported claims | Reduce scope and tie claims to evidence |

### Reviewability Check

Another engineering team should be able to inspect the risk evidence and understand major engineering, governance, operational, and release concerns.

## Traceability Example

**Purpose:** Connects requirements to tasks, GitHub Issues, Pull Requests, tests, operational findings, and release evidence.

**Typical repository location:** `/docs/planning/traceability.md`

### Engineering Guidance

Traceability evidence should remain discoverable through the repository README, linked engineering artifacts, pull requests, and release evidence when appropriate.

Traceability should make it possible for another engineering reviewer to understand how important system behavior, risks, and release claims were validated.

### Traceability Table

| Requirement ID | Requirement Summary | GitHub Issue(s) | Task / Work Item | PR / Commit Evidence | Test / Validation / Operational Evidence | Release Evidence | Status |
|---|---|---|---|---|---|---|---|
| FR-001 | Student can submit a support request | #1 | WB-001: Build request form and validation | PR #5 | TC-001, TC-002, validation screenshot | Release Notes v0.1 | In progress |
| FR-002 | Staff user can view submitted requests | #2 | WB-004: Build request list view | PR #7 | TC-003, review notes | Release Notes v0.1 | Planned |
| FR-003 | Staff user can update request status | #3 | WB-006: Add manual status update | PR #9 | TC-004, defect D-002 | Release Notes v0.1 | Planned |
| NFR-001 | Important failures are diagnosable | #12 | WB-010: Add basic runtime logging | PR #14 | Log screenshot, operational review note | Known limitation / Release Notes | Open |

### Traceability Notes

- Every major requirement should eventually connect to issues, tasks, branches, pull requests, reviews, tests, release evidence, and operational findings when applicable.
- Every completed feature should connect to review, test, and release evidence.
- If a requirement is deferred, explain why and where that decision is recorded.
- AI-assisted implementation work should remain traceable through issues, pull requests, review evidence, AI-use logs, validation evidence, and release documentation when applicable.

### Deferred / Removed Requirements

| Requirement ID | Decision | Reason | Evidence / Decision Record |
|---|---|---|---|
| FR-999 | Deferred | Live student information system integration requires private data and institutional approval | Scope OOS-001; Risk R-005; ADR-003 |
| AI-001 | Deferred | AI category suggestion lacks sufficient validation and governance evidence for Cycle 1 | Scope D-001; AI-use log; Risk R-003 |

### Traceability Gaps

| Gap | Impact | Owner | Next Action |
|---|---|---|---|
| NFR-001 has limited evidence | Runtime failures may be hard to explain during release review | Example: Quality Lead | Add logging task and operational evidence |
| FR-003 lacks reviewed PR | Status workflow cannot be claimed as release-ready yet | Example: Development Lead | Complete PR and validation evidence |

### Reviewability Check

Another engineering team should be able to inspect the traceability evidence and understand how requirements evolved into implemented, reviewed, tested, and released engineering behavior.

## Release Notes

### Purpose

This document summarizes what is included in the current release candidate.

Release notes should help reviewers understand what changed, what is supported, what is deferred, and what evidence exists.

### Release Summary

| Item | Value |
|---|---|
| Release Name / Version | |
| Release Date | |
| Cycle | Cycle 1 / Cycle 2 |
| Release Owner | |
| Status | Draft / Ready for Review / Released |

### Included Capabilities

| Capability | Related Requirement | Evidence |
|---|---|---|
| | | |

### Major Changes

| Change | Reason | Evidence |
|---|---|---|
| | | |

### Verification Summary

| Evidence Type | Location |
|---|---|
| Test evidence | |
| Review evidence | |
| AI verification evidence | |
| Release readiness review | |

### Deferred Items

| Deferred Item | Reason | Future Action |
|---|---|---|
| | | |

### Engineering Standard

Release notes should make release claims understandable and reviewable.

## Known Limitations

### Purpose

This document records known limitations, unresolved risks, deferred items, and constraints.

Known limitations are engineering evidence.

They help reviewers understand what the team is not claiming.

### Known Limitations

| ID | Limitation | Impact | Related Evidence | Status |
|---|---|---|---|---|
| LIM-001 | | | | Open |

### Accepted Risks

| Risk | Reason Accepted | Mitigation | Owner |
|---|---|---|---|
| | | | |

### Deferred Work

| Item | Reason Deferred | Required Evidence Before Future Release |
|---|---|---|
| | | |

### Review Questions

- What are we not claiming?
- What risks remain?
- What should reviewers know before trusting the release?
- What requires future work?

### Engineering Standard

Honest limitations increase trust.

Hidden limitations create engineering risk.

## Release Readiness Review

### Purpose

This document records release readiness findings.

The goal is determining whether engineering claims can be defended.

A release review is not asking:

> Does it work?

A release review asks:

> Can we defend releasing it?

### Release Review Entries

| Date | Release Candidate | Findings | Risks Remaining | Actions | Reviewer |
|---|---|---|---|---|---|

### Review Areas

Examples:

- requirements evidence
- testing evidence
- risk mitigation
- AI governance
- operational readiness

### Review Questions

- Is evidence sufficient?
- Are risks understood?
- Are limitations documented?

### Engineering Standard

A demo is not operational proof.

## Test Plan

### Purpose

This document defines what will be tested and when.

Testing plans should remain lightweight and adaptable.

### Testing Activities

| Area | Validation Activity | Owner | Status |
|---|---|---|---|
| | | | |

### Testing Schedule

Examples:

- feature completion
- integration milestones
- release preparation

### Assumptions

Document assumptions that influence testing.

### Review Questions

- Is testing aligned with requirements?
- Are responsibilities clear?
- Are important milestones covered?

### Engineering Standard

Strong testing is planned rather than reactive.