CSE 416-03 Software Engineering — Syllabus (Fall 2026)

Instructor: Shuai Mu

Class: MW 2:00–3:20pm, NCS 120 (Aug 24 – Dec 7)

Office hours: MW 1–2pm, NCS 351. Optional: You can book a GCal meeting to lock in a time to talk.

Piazza: https://piazza.com/stonybrook/fall2026/cse41603

Overview

This course introduces the basic concepts and modern tools of software engineering. The emphasis is on building reliable, maintainable software: requirements, design, implementation, testing, project management, documentation, and the human side of working on a team.

We learn this mainly by doing it. You will work in a team on a semester-long project of your own choosing and carry it through its complete lifecycle (proposal, design, implementation, testing, and release), checked at six milestones. There is no required domain, architecture, or tech stack. Propose something you actually want to build.

A few lectures cover the practices that matter most today. The rest of the semester is milestone reviews and student-led presentations. AI coding tools are encouraged; using them well is part of the course. There are no exams.

Prerequisites

Officially: C or higher in CSE 316; U4 standing; CSE major.

Informally: you should be almost ready to work as a software engineer. You should already be comfortable with at least one scripting language (for example JavaScript or Python) and at least one strongly typed language (for example Java, C/C++, or Rust), and you should be able to pick up a new language or library from online resources without extra tutoring.

CSE 305, CSE 333, CSE 336, CSE 337, and CSE 356 are not required. Depending on what you build, you may need to spend extra time picking up databases, UI, or a new language. That is expected.

Getting started

- Join Piazza. Course announcements go there.
- Form a team during the Aug 26 and Aug 31 sessions. It is your responsibility to join a team.
- Lock in a project idea you actually want to build. There is no assigned domain.
- Presentation topics will be assigned after teams are set.

Share a GitHub (or similar) repository with the course staff once your team is formed. We will give details in class.

Learning outcomes

By the end of the course, you should be able to:

- Write and evaluate requirements, and turn them into a design (diagrams, specs, or whatever notation fits the project).
- Choose a process that a small team can actually follow on a fixed timeline.
- Design an architecture, justify the trade-offs, and implement it.
- Build, test, debug, and iteratively improve a non-trivial system with your teammates, using version control, code review, and some form of CI.
- Use AI coding tools productively, and explain both the code and the decisions.
- Communicate those decisions in writing and in a presentation.

Project

The project is the course. Work in a team. Pick something you actually want to ship by December. A web app or a desktop app are both fine. We will not assign you a topic, a stack, or an architecture.

If you would rather not invent a product from scratch, you can join a **staff project** and work with the instructor. There are two tracks, both large systems projects:

- **Chiral Network — a blockchain.** Last year's project, continued. Resource sharing (storage, compute, bandwidth, ...) and hosted functions on the chain, in the spirit of the Internet Computer. Not just a token.
- **Cloud database** — a cloud database in the spirit of Supabase: auth, storage, a query API, Postgres-shaped and meant to be a real backend for apps.

Same milestones and grading as everyone else. Say so when you form teams; seats are limited. These are a good fit if you like systems. They are not a shortcut.

It needs to be (designed to be) a real product with **real users**: accounts, something people can log into and actually use, and enough functionality that it is a semester of work. A thin utility page does not count, even if it technically has visitors. A campus map of Stony Brook, a torrent search site, or any similar one-purpose tool page is too easy.

A few examples about what does count: something like Mint or Monarch (money management), a shopping site, or a Piazza-like discussion system, or a complex enough game. If you are unsure whether an idea is complex enough, ask before you lock it in.

Not allowed. Anything with morality concerns is disallowed. That includes gambling, adult content, and prediction markets (for example Polymarket-style betting). If you are unsure, ask.

What we will look for is that you ran a real software process: you scoped the work, designed it, built it, tested it, and can explain the choices. Using AI to move faster is encouraged; using AI as a substitute for understanding is not.

Each milestone has an in-class review. Teams are graded on the day they present. Bring a short demo or walkthrough of what you have, and have the artifacts below ready in the

repository.

Milestones

Every milestone review is about 10 minutes. Cover (1) what you finished, (2) the design choices and why, and (3) how the team (including AI) worked together in this phase.

Milestone 1 — Proposal and Requirements (10%). Problem, users, why this is a semester of work (and not a weekend with ChatGPT), related systems, initial scope, and who does what. Artifacts: written proposal, functional and non-functional requirements, initial user stories or equivalent, team roles, rough architecture sketch.

Milestone 2 — Design and Setup (10%). Architecture, tech stack, and a demo of whatever is already running. Artifacts: design document (diagrams as needed), stack justification, repository with a CI skeleton, a minimal prototype.

Milestone 3 — MVP (10%). Live demo of the core feature end to end, plus your testing approach so far. Artifacts: working MVP, updated design docs, an initial test suite or a documented test plan.

Milestone 4 — Feature-Complete (10%). Demo of the full planned feature set, trade-offs you made, and the plan for testing and integration. Artifacts: complete implementation of planned features, documented test plan, updated docs.

Milestone 5 — Testing and Integration (10%). Integration results, bugs you found and fixed, deployment plan. Artifacts: tests and results, a deployed (or otherwise runnable) version, draft user/developer docs.

Milestone 6 — Final Product (30%). Final demo and a short retrospective: what worked, what did not, how AI was used across the project. Artifacts: working product, cleaned-up repository and docs, final report including the retrospective.

Dec 7 — Poster session. Last day of class. Show the finished project.

Grading

- **Course project (80%)**, in six milestones: 10% + 10% + 10% + 10% + 10% + 30%.
- **Class presentation (20%)**, one 30-minute talk per group.

No exams. Milestone grades consider the quality of what the team delivers and each student's contribution. We will ask you to assess your teammates at milestones.

If you find a grading error, tell us. Do not negotiate for extra points or extra work.

Class presentations

Each team gives one **30-minute** presentation on a software-engineering topic. All members are expected to speak (about four or five people). You can split one big story across the team, or each person can tell a different smaller story on the same topic.

The talk should be a **story**, not a lecture. Pick a real use case: how a company, a well-known incident, or a concrete project actually practices this topic. What do they do, what is it for, and what happened? Interesting failures count. We want to hear how the work is actually done, not a recap of the textbook.

For example, if the topic is testing, do not spend the time explaining what a unit test or a regression test is. Tell us how some team/company actually tests: how they organize it, what they test for, a bug testing caught (or failed to catch), and why that process looks the way it does.

Strongly discouraged, you would get a low grade if you do this: going over concepts, taxonomies, or "what is X" slides. Assume the class can look those up. We are not looking for explanations of the terms; we are looking for stories.

Send slides at least 72 hours before your slot.

Date	Topic
Sep 21	Presentation 1: User Experience (UI/UX and user study)
Sep 23	Presentation 2: Software Architecture and Design Patterns
Oct 5	Presentation 3: Prompt Engineering and AI-Assisted Development
Oct 7	Presentation 4: API Design and Documentation
Oct 14	Presentation 5: CI/CD and DevOps
Oct 26	Presentation 6: Testing
Oct 28	Presentation 7: Debugging
Nov 9	Presentation 8: Software Security and Secure Coding Practices
Nov 11	Presentation 9: Software Maintenance, Technical Debt and Refactoring
Nov 23	Presentation 10: Responsible and Ethical Tech

{:.table-striped}

AI

AI is encouraged; use it as much as you can. Autocomplete tools (Copilot and similar) and agentic tools (Cursor, Claude Code, and similar) are all fine on the project.

Ground rules:

- **Understand before you ship.** Do not commit code you cannot explain to a teammate.

- **Write your own tests.** Do not outsource verification to the same tool that wrote the code.
- **Disclose AI use.** Note where it contributed, as you would cite any other source.
- **Keep the important decisions human.** Use AI to implement, not to decide what to build or why.

AI does not reduce your responsibility for correctness, security, or quality.

Academic integrity

This is a team project course. Work with your teammates; that is the point. Do not copy another team's project, presentation, or writeup, and do not claim someone else's work as yours. Public libraries, Stack Overflow, papers, and similar sources are fine if you cite them. Ask if you are unsure.

Violations lead to an F and a report to the university. Enrollment in this class means you have accepted this policy.

University statement. Each student must pursue their academic goals honestly and be personally accountable for all submitted work. Representing another person's work as your own is always wrong. Faculty is required to report any suspected instances of academic dishonesty to the Academic Judiciary. For more comprehensive information on academic integrity, including categories of academic dishonesty, please refer to the academic judiciary website.

Student Accessibility Support Center

If you have a physical, psychological, medical, or learning disability that may impact your course work, please contact the Student Accessibility Support Center, Stony Brook Union Suite 107, (631) 632-6748, or at sasc@stonybrook.edu. They will determine with you what accommodations are necessary and appropriate. All information and documentation is confidential.

Critical Incident Management

Stony Brook University expects students to respect the rights, privileges, and property of other people. Faculty are required to report to the Office of Student Conduct and Community Standards any disruptive behavior that interrupts their ability to teach, compromises the safety of the learning environment, or inhibits students' ability to learn. Further information about most academic matters can be found in the Undergraduate Bulletin, the Undergraduate Class Schedule, and the Faculty-Employee Handbook.

The class calendar is on the schedule page.