NEW JERSEY INSTITUTE OF TECHNOLOGY · COURSE SYLLABUS

CS 690

AI-Assisted Software Engineering

Fall 2026

INSTRUCTOR

Mohit Dale

Office Hours: TBD

mohit.dale@njit.edu · md685@njit.edu

LOCATION & CONTACT

GITC 4406

Office Phone: (973) 596-2655

Expected effort: 8–12 hrs/week outside lecture

Course Description

Modern generative artificial intelligence tools are astonishingly effective at generating code from natural-language specifications. The software engineering industry is rapidly adopting these tools to improve engineers' productivity: rather than writing all of their code themselves, many engineers are now effectively "team leaders," managing a "team" of artificial intelligence tools. In this course, students will gain hands-on experience applying such tools to realistic software engineering tasks in a semester-long team project.

This course focuses on a team-based, industrial-scale software project and spans the software lifecycle end to end: requirements analysis and specification, project planning, software architecture and design, implementation, testing, and deployment, along with the associated documentation.

The course combines rigorous classroom study of the underlying principles — large language models, agent architectures, and specification and verification — with a studio component in which teams build, review, and maintain a non-trivial software system using AI coding assistants. Topics include agents; requirements elicitation and specification in the AI era; AI code generation and techniques for ensuring that AI-generated code is correct; and how traditional software engineering practices, including code review, testing, and static analysis, must evolve to support AI-assisted development.

This course is designed for students who intend to lead engineering teams, conduct research at the intersection of software engineering and machine learning, or shape the technical practices of organizations adopting generative AI. It is a demanding course with a substantial project component.

Course Objectives

By the end of the semester, this course aims to prepare students to:

- Reason rigorously about when, why, and how generative AI tools accelerate or degrade software engineering outcomes, grounded in empirical evidence rather than vendor claims.

- Operate AI coding assistants fluently across the full development lifecycle: requirements, design, implementation, testing, review, deployment, and maintenance.
- Design and orchestrate software engineering agents, including single-agent loops (ReAct, Reflexion) and multi-agent systems, using industry frameworks such as LangGraph, the Anthropic SDK, and related tooling.
- Elicit, document, and communicate requirements and specifications that are precise enough to drive reliable AI code generation and robust enough to survive iteration.
- Assure the correctness of AI-generated code using a layered arsenal: type systems, tests, property-based testing, static analysis, dynamic analysis, and, where warranted, formal verification.
- Adapt classical software engineering practices, including code review, architecture, refactoring, and DevOps, to the realities of mixed human/AI authorship.
- Critically evaluate the security, safety, legal, and ethical dimensions of AI-assisted software engineering, including prompt injection, data leakage, licensing, attribution, and the changing nature of engineering labor.
- Lead a small engineering team in delivering a working software system, exercising the judgment that distinguishes a senior engineer from a prompter.

Student Learning Outcomes

Upon successful completion of this course, a student will be able to do each of the following:

Knowledge — what students will know

1. Explain the architecture and training pipeline of contemporary code-generation LLMs, including pretraining corpora, instruction tuning, RLHF/RLAIF, and inference-time techniques such as chain-of-thought and self-consistency.
2. Summarize the state of the art in AI coding assistants and agents, and situate a given tool (e.g., GitHub Copilot, Cursor, Claude Code, Aider, Devin, SWE-agent) within a principled taxonomy.
3. Articulate the strengths and limitations of widely used code-generation benchmarks (HumanEval, MBPP, SWE-bench, LiveCodeBench) and reason about external validity.
4. Characterize the threat model of prompt injection, indirect prompt injection, model poisoning, and supply-chain risks specific to AI-assisted development.

Skills — what students will be able to do

5. Author structured, testable requirements and specifications suitable for consumption by AI tools, using user stories, acceptance criteria, type signatures, invariants, and formal contracts.
6. Design effective prompts, system prompts, and retrieval strategies for code-related tasks, and systematically evaluate their quality through A/B studies and regression suites.
7. Implement an agentic workflow, including tool use, memory, error recovery, and human-in-the-loop checkpoints, from scratch using a production LLM API.
8. Construct a layered verification and validation pipeline for AI-generated code, integrating static analysis, type checking, unit and property-based tests, fuzzing, and targeted formal reasoning.

9 Conduct and receive professional-quality code review of AI-generated contributions, distinguishing issues of correctness, security, maintainability, and style.

Dispositions — how students will engage professionally

10 Exercise calibrated trust: assess AI output with appropriate skepticism, and document decisions, failure modes, and residual risk.

11 Attribute authorship honestly, in accordance with course policy, institutional standards, and the norms of the profession.

12 Communicate technical tradeoffs in writing, in code review, and in presentations clearly enough that both engineers and non-technical stakeholders can make informed decisions.

Required & Recommended Materials

Textbooks (Optional)

There is no single textbook that is current on this subject. The following texts, plus a weekly reading packet of primary research papers and industry reports, form the reading list. All texts and weekly readings are **optional**: they are recommended to deepen your understanding, but none are required purchases.

- **Sommerville, I.** *Software Engineering*, 11th ed. (Pearson) — reference text for classical SE material.
- **Kleppmann, M.** *Designing Data-Intensive Applications* (O'Reilly) — reference for systems and architecture chapters.
- **Jurafsky, D. & Martin, J. H.** *Speech and Language Processing*, 3rd ed. draft (freely available online) — selected chapters on transformers and language modelling.

Software & Accounts

Students must have working access to the following by the end of Week 1.

- An AI coding assistant with agent capabilities (Claude Code, Cursor, Aider, or equivalent). The canonical studio tool will be announced at the start of the term.
- Programmatic access to a frontier LLM API (e.g., Anthropic, OpenAI) sufficient for agent development.
- A GitHub account enrolled in the course's GitHub Classroom organization.
- A local development environment with Python ≥ 3.11, a statically typed language toolchain of the student's choice, Docker, and a modern editor.
- Static analysis and testing tools such as mypy or pyright, ruff, pytest, Hypothesis (property-based testing), and ESLint/TypeScript for JS/TS work.

Weekly Schedule

1 Introduction: The Paradigm Shift in Software Engineering

- A brief economic and technical history of SE automation: from compilers to IDEs to copilots
- The "engineer as team lead" thesis and its critiques
- Course logistics, project formation, and studio norms

READINGS

- Brooks, "No Silver Bullet: Essence and Accident in Software Engineering" (1987)
- Willison, "2025: The Year in LLMs" (2025)
- Anthropic, "How AI Assistance Impacts the Formation of Coding Skills" (2025)
- Sommerville Ch. 1–2 (skim)

2 Foundations: Large Language Models for Code

- Transformers in one lecture: tokenization, attention, pretraining objectives
- From Codex to frontier coding models: instruction tuning, RLHF, RLAIF
- Benchmarks and their limitations: HumanEval, MBPP, SWE-bench, LiveCodeBench, contamination

READINGS

- Chen et al., "Evaluating Large Language Models Trained on Code" (Codex, 2021)
- Jimenez et al., "SWE-bench: Can Language Models Resolve Real-World GitHub Issues?" (2024)
- Jurafsky & Martin, selected transformer chapters

3 Prompt Engineering, Context, and Retrieval

- Zero-shot, few-shot, chain-of-thought, and self-consistency prompting
- System prompts, structured outputs, tool/function calling, JSON mode
- Retrieval-Augmented Generation over repositories; embedding models and code search

MILESTONE · PROJECT PROPOSAL DUE

READINGS

- Wei et al., "Chain-of-Thought Prompting Elicits Reasoning" (2022)
- Lewis et al., "Retrieval-Augmented Generation for Knowledge-Intensive NLP" (2020)
- Anthropic, "Prompt Engineering Overview" (current documentation)
- Anthropic, "Effective Context Engineering for AI Agents" (2025)

4 Requirements Elicitation and User Discovery in the AI Era

- Classical requirements engineering: stakeholders, user stories, use cases, IEEE 830
- Customer discovery: interviewing real users, "Mom Test" questioning, and AI-generated "synthetic users" and their limits
- AI-assisted elicitation: interview transcription, summarization, ambiguity detection
- When natural language isn't enough: moving toward executable specifications

READINGS

- Sommerville Ch. 4 (Requirements Engineering)
- Fitzpatrick, *The Mom Test* (selections)
- Nielsen Norman Group, "Synthetic Users: If, When, and How to Use AI-Generated 'Research'"
- "Analysis of LLMs vs Human Experts in Requirements Engineering" (2025)
- Cohn, *User Stories Applied* (selections)

5 Specification, Intent, and Design by Contract

- Formal vs. informal specifications; pre/postconditions; invariants
- Types as specifications; dependent and refinement types in practice
- Writing development specs: design docs precise enough for AI implementation
- Spec-driven code generation; the prompt as specification

READINGS

- Meyer, "Applying Design by Contract" (1992)
- Parnas, "On the Criteria to Be Used in Decomposing Systems into Modules" (1972)
- "Design Docs at Google" (industrialempathy.com)
- "When Prompts Go Wrong: Evaluating Code Model Robustness to Ambiguous, Contradictory, and Incomplete Task Descriptions" (2025)

6 Building with LLMs I: Frontend Development

- Completion vs. function-level synthesis vs. repository-level generation
- Iterative refinement and self-repair workflows in production tools (Claude Code, Cursor, Copilot, Aider)
- Prototyping ("vibe coding") vs. engineering: when each is appropriate
- Generating, styling, and iterating on a working UI with an agentic tool

LAB · IN-CLASS FRONTEND BUILD (BRING LAPTOPS)

READINGS

- Hashimoto, "Vibing a Non-Trivial Ghostty Feature" (2025)
- "Good Vibrations? A Qualitative Study of Co-Creation, Communication, Flow, and Trust in Vibe Coding" (2025)
- Olausson et al., "Is Self-Repair a Silver Bullet for Code Generation?" (2024)
- Chen et al., "Teaching Large Language Models to Self-Debug" (2023)

7 Agents and Agentic Software Engineering

- Agent architectures: ReAct, Reflexion, Plan-and-Solve, tree search
- Tools, memory, and human-in-the-loop control
- Multi-agent orchestration (LangGraph, AutoGen, CrewAI); autonomous engineers (SWE-agent, OpenHands, Devin)

MILESTONE 1 REVIEW · IN CLASS, ATTENDANCE MANDATORY

READINGS

- Yao et al., "ReAct: Synergizing Reasoning and Acting in Language Models" (2023)
- Shinn et al., "Reflexion: Language Agents with Verbal Reinforcement Learning" (2023)
- Yang et al., "SWE-agent: Agent-Computer Interfaces Enable Automated Software Engineering" (2024)

8 Midterm + Testing with AI: TDD and Test Quality

- In-class midterm examination
- Test-driven development with LLM agents: tests as executable specifications, red-green-refactor loops
- AI-generated unit, integration, and property-based tests
- Mutation testing and test-adequacy metrics in AI-assisted pipelines

READINGS

- Shore, "Test-Driven Development" (*The Art of Agile Development*)
- "TDFlow: Agentic Workflows for Test Driven Development" (2025)
- "Software Testing with Large Language Models: An Interview Study with Practitioners" (2025)
- Schäfer et al., "Empirical Evaluation of Using LLMs for Automated Unit Test Generation" (2023)
- Claessen & Hughes, "QuickCheck: A Lightweight Tool for Random Testing" (2000)

9

Building with LLMs II: Backend, Architecture, and Refactoring

- Backend services, data models, and APIs with LLM agents
- AI in architectural decisions: ADRs, tradeoff analysis, quality attributes
- Refactoring at scale with AI: codemods versus conversational edits
- Legacy comprehension and modernization workflows

LAB · IN-CLASS BACKEND BUILD SESSION

READINGS

- "From Code to Correctness: Closing the Last Mile of Code Generation with Hierarchical Debugging" (2024)
- "Towards Understanding the Characteristics of Code Generation Errors Made by Large Language Models" (ICSE 2025)
- Bass, Clements & Kazman, *Software Architecture in Practice* (selections)
- Fowler, *Refactoring*, 2nd ed. (selections)

10

Ensuring Correctness: Static Analysis, Types, and Verification

- Linters and type checkers as a safety net for AI output
- Abstract interpretation and symbolic execution, in brief
- SMT solvers and AI-assisted proof engineering (Dafny, Lean, Verus)

READINGS

- GitHub, "Why AI Is Pushing Developers Toward Typed Languages" (2025)
- Cousot & Cousot, "Abstract Interpretation" (1977), survey summary
- First et al., "Baldur: Whole-Proof Generation and Repair with LLMs" (2023)
- "PredicateFix: Repairing Static Analysis Alerts with Bridging Predicates" (2025)

11

Security and Risks of AI-Generated Code

- Vulnerability classes overrepresented in AI output (CWE patterns, OWASP Top 10)
- Prompt injection and indirect prompt injection; data exfiltration; secret handling
- Supply-chain and dependency risks; governance at scale

READINGS

- Pearce et al., "Asleep at the Keyboard? Assessing the Security of GitHub Copilot's Code Contributions" (2022)
- Perry et al., "Do Users Write More Insecure Code with AI Assistants?" (2023)
- Greshake et al., "Not What You've Signed Up For: Compromising Real-World LLM-Integrated Applications with Indirect Prompt Injection" (2023)
- "Smoke and Mirrors: Jailbreaking LLM-based Code Generation via Implicit Malicious Prompts" (2025)

12

Code Review in the AI Era

- Reviewing AI-generated code: cognitive load, calibration, and anti-patterns
- AI-assisted human review: diff summarization, risk scoring, reviewer fatigue
- Pull-request workflows when agents open PRs

MILESTONE 2 REVIEW · IN CLASS, ATTENDANCE MANDATORY

READINGS

- Bacchelli & Bird, "Expectations, Outcomes, and Challenges of Modern Code Review" (2013)
- "A Causal Perspective on Measuring, Explaining and Mitigating Smells in LLM-Generated Code" (2025)
- Recent research on LLM-assisted review (LMS packet)

13

Deployment in Practice: CI/CD, Cloud, and AIOps

- Containerized builds and one-command deployment of the team project
- CI/CD pipelines with AI gates: code review, test generation, security checks
- Cloud deployment patterns: serverless backends, managed frontend hosting, continuous delivery
- Monitoring and incident response with LLMs: log analysis, root-cause assistance; infrastructure as code and AI-assisted SRE

LAB · EACH TEAM SHIPS A LIVE BUILD

READINGS

- AWS Amplify "Getting Started" tutorial
- AWS Lambda + API Gateway tutorial
- Beyer et al., *Site Reliability Engineering*, "Monitoring Distributed Systems" chapter
- Recent AIOps research and postmortems (LMS packet)

14

Professional, Ethical, and Legal Dimensions

- IP, copyright, licensing, and provenance (training data, model outputs, end products)
- Attribution, authorship, and academic integrity in an AI-authored world
- Labor economics, productivity claims, and the future of the software profession

READINGS

- Weidinger et al., "Ethical and Social Risks of Harm from Language Models" (DeepMind, 2021)
- "An LLM Agentic Approach for Legal-Critical Software: A Case Study for Tax Prep Software" (2025)
- Current scholarship on AI and copyright (LMS packet)

15

Project Presentations & Future Directions

- Team final presentations with live demonstration (20 min + 10 min Q&A)
- Survey of research frontiers: self-improving systems, autonomous maintenance, specification synthesis
- Course retrospective

FINAL DELIVERABLES DUE · PRESENTATIONS, ATTENDANCE MANDATORY

READINGS

- Selected position papers on the future of software engineering (LMS packet)

The Semester Project

The project is the spine of this class. Students work in teams of three or four to design, implement, verify, and maintain a non-trivial software system primarily through AI-assisted development. Example system classes include, but are not limited to: a domain-specific static analyzer, a small database, a collaborative editor, a distributed job scheduler, a compiler front-end, a data-pipeline framework, a web application with non-trivial business logic, or an AI agent built on top of a production model.

Deliverables

WEEK 3 · PROPOSAL

A two-page document defining scope, success criteria, and the verification strategy — approved before work begins.

WEEK 7 · MILESTONE 1

Architecture and requirements review; draft specifications; prompts and agent harness checked in; test skeleton in place.

WEEK 12 · MILESTONE 2

Mid-project demonstration; end-to-end feature slice working; comprehensive CI with static analysis, tests, and security gates; risk register.

WEEK 15 · FINAL

Working system with a tagged release; a 12–15 page technical report; a 20-minute live demo; complete provenance log of AI contributions and their reviews.

Individual Accountability

Although deliverables are produced by teams, each student must submit a signed contribution statement with each milestone, documenting personal authorship, AI-assisted authorship, and review responsibilities. A confidential teammate evaluation accompanies the final submission; persistent free-riding will adjust individual grades.

Artifact Requirements

To receive full credit, the final artifact must satisfy the following minimum standards:

1. A machine-checkable specification layer (types, invariants, or contracts) sufficient to express the system's core safety properties.
2. A test suite (unit, integration, and property-based) with ≥ 80% line coverage and a documented mutation score.
3. A clean static-analysis gate (chosen linter and type checker, zero high-severity findings).
4. A documented security review, including a threat model and remediation notes.
5. A reproducible build and a one-command deployment, containerized where applicable.
6. A provenance ledger: every AI-assisted commit annotated with the tool, prompt summary, and review actions taken.

> **The provenance ledger is the single most important deliverable of the course.**

Assessment & Grading

Grades are computed on a 100-point scale from the weighted components below. All deliverables are submitted via the LMS unless otherwise specified.

Semester project	35%
Assignments	25%
Midterm	20%
Final exam	20%

Course Policies

Academic Integrity and the Use of AI Tools

This is a course about the responsible use of AI tools for software engineering. The ordinary prohibitions on unauthorized collaboration or misrepresentation of authorship apply in full, but they are operationalized carefully in this setting. The following policy is binding; students are expected to read it with the same care they would read a software license.

Permitted, expected, and required uses of AI

AI coding assistants are expected tools in this course. Their use is permitted on every assignment unless explicitly marked "no AI" on the assignment handout (a small number of in-class exercises will be so marked to isolate a learning objective).

On every assignment, students must record and submit a **provenance log**: which tools were used, what prompts were issued (summary form acceptable for long sessions), and what review actions were taken on the generated output. Submitting AI-generated code without review is a form of misrepresentation and will be treated as a violation.

On the midterm and in-class examinations, AI tools are **not permitted**. Students are expected to be able to demonstrate mastery of the material independently.

On individual written work (including the professional-ethics position paper and the final reflective synthesis), AI tools may be used to polish prose but not to generate substantive argument. Any such polishing must be disclosed.

Violations — including submitting unreviewed AI output, fabricating provenance logs, or representing teammates' work as one's own — are referred to the Dean of Graduate Studies under the institution's academic-integrity policy. Penalties include grade reduction, course failure, and, for egregious cases, dismissal from the program.

Attendance

Attendance at lectures is expected. Students anticipating a conflict should notify the instructor in advance where possible. Attendance at the two milestone reviews and the final presentation is **mandatory**; unexcused absence results in a score of zero for that milestone.

Changes to the Syllabus

Given the rapid evolution of this field, the instructor reserves the right to update readings, tool choices, and the order of topics during the term. Substantive changes — in particular to grading weights or due dates — will be announced in class and on the LMS at least one week in advance.

A Small Note from Me

The premise of this course is that AI tools are changing what it means to be a software engineer — not by eliminating the craft, but by elevating where the craft is practiced. The students who will thrive in the coming decade are those who can specify problems precisely, orchestrate tools effectively, verify results rigorously, and take honest responsibility for the code they ship. My goal for this class is that each of you leaves this course as exactly that kind of engineer.

Please come to office hours. Ask questions. Software engineering is, at its best, a deeply collaborative intellectual pursuit.