Incorporating Generative AI into Software Engineering Courses

Chris Bohn
bohn@unl.edu

Bonita Sharif
bsharif@unl.edu

School of Computing
University of Nebraska-Lincoln

Background

- College of Engineering's Prairie initiative includes "ensure every student graduates with the AI competencies employers demand"
- Fall 2025: GenAI activities added to several courses in SE major, and CS1/CS2, SE course for CS/CE majors
- The School of Computing has drafted an AI policy that addresses
  - AI as a productivity tool
  - AI as a software system to be studied
  - Academic integrity
  - Instructor discretion

SOFT 160

- CS1 equivalent course for SE major
- Fall 2023, Fall 2024: No AI allowed except CodeHelp.app
- Fall 2025: Some activities & assignments include AI

SOFT 160 Practice Problem Set 2
Code Readability, Intro to CodeHelp

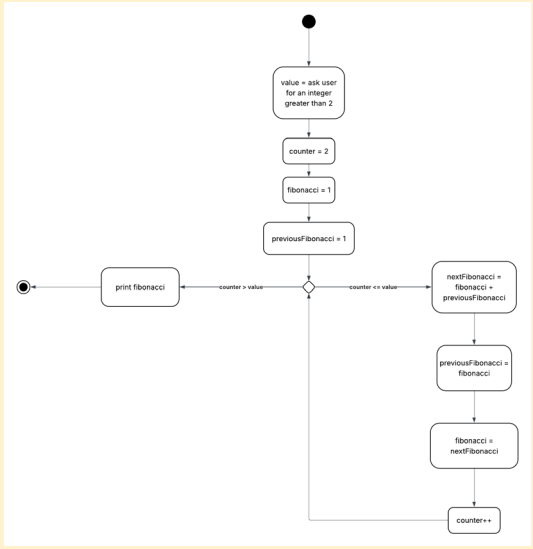
- Intentionally unreadable code – ask ChatGPT (or other LLM) what it does
  - Four seeded bugs
  - Poor names, unnecessarily-complex divisibility check

- Readable version – ask ChatGPT what it does
  - Is the description better?
  - What bugs were found?

- Ask CodeHelp.app for help (“good” question suggested)
  - Were additional bugs found?
  - What suggestions were offered?

SOFT 160 Practice Problem Sets 6, 10, 12
Diagrams to Code

- Week 6: UML activity diagram for problem with conditionals
  - LLMs preferred chained conditionals or nested conditionals instead of distinct if statements



- Week 10: UML activity diagram for problem with a loop
  - Faithful loops
  - ¾ saw unspecified do loop for user input

```java
import java.util.Scanner;

public class FibonacciFlowchart {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        int value;

        // Ask user for an integer greater than 2
        do {
            System.out.print("Enter an integer greater than 2: ");
            value = scanner.nextInt();
        } while (value <= 2);

        int counter = 2;
        int fibonacci = 1;
        int previousFibonacci = 1;

        while (counter <= value) {
            int nextFibonacci = fibonacci + previousFibonacci;
            previousFibonacci = fibonacci;
            fibonacci = nextFibonacci;
            counter++;
        }

        // Print the nth Fibonacci number
        System.out.println("Fibonacci number at position " + value + " is: " + fibonac
    }
```

- Week 12: UML class diagram
  - Faithful code stubs

SOFT 160 Practice Problem Set 8
Spec to Code

- Existing question
  - Bad spec for “faster” arithmetic – *Design a test suite*
  - Partial implementation provided – *would your test suite cover all paths?*
- Leads to in-class discussion about non-testing V&V to catch problems early

- New follow-up question: Ask LLM to implement the algorithm
  - LLMs never flagged that the spec cannot be correctly implemented
  - All implementations deviated from the spec; about ¾ produced functionally correct (sometimes slower) arithmetic

SOFT 160 Homework 6 Extra Credit
Vibe Coding

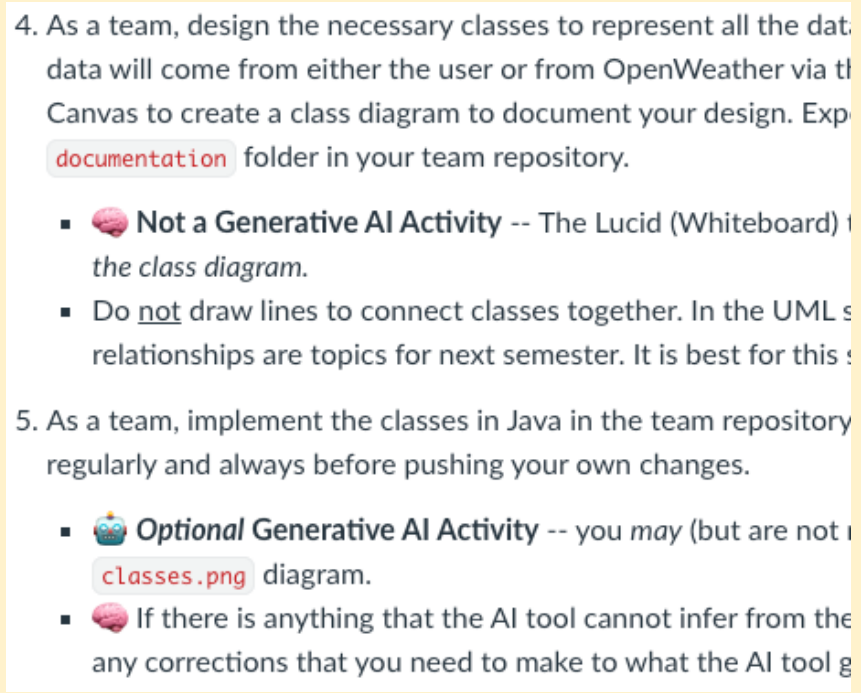
- Test Driven Development assignment
  - Part 1: design test suite
  - Part 2: make your test suite pass
  - Part 3: make instructor's test suite pass
- Extra credit Part 4 (overlaps with two other assignments)
  - Video demonstration
  - Have LLM implement the spec, one requirement at a time
  - Use Part 1's test suite to verify solution
  - Don't go to next requirement until all tests for current requirement pass
- Compare code quality from Part 2, Part 4

SOFT 160 Homework 6 Extra Credit
Vibe Coding - student comments

- “If vibe coding was made a part of the course near the beginning, it could help people understand coding a bit faster”
- “Reinforced importance of reviewing generated code and testing thoroughly … required active engagement to ensure correctness”
- “Easy when things are going well … it can take a while to fix its problems”
- “I do not get the same satisfaction when passing tests as when making my own code pass tests”

SOFT 160 Final Project

- Early in project, student teams prepare class diagram then implement it in Java


4. As a team, design the necessary classes to represent all the dat
data will come from either the user or from OpenWeather via t
Canvas to create a class diagram to document your design. Exp
`documentation` folder in your team repository.

- 🧠 **Not a Generative AI Activity** -- The Lucid (Whiteboard)
the class diagram.
- Do not draw lines to connect classes together. In the UML s
relationships are topics for next semester. It is best for this

5. As a team, implement the classes in Java in the team repository
regularly and always before pushing your own changes.

- 🤖 ***Optional*** **Generative AI Activity** -- you *may* (but are not
`classes.png` diagram.
- 🧠 If there is anything that the AI tool cannot infer from the
any corrections that you need to make to what the AI tool g

- Student teams had the option to prepare class diagram manually and have LLM generate Java class stubs
- At least one team did *not* exercise the option
- Fewer penalties for disagreement between class diagram and Java code than in earlier years

SOFT 160 – Lessons Learned

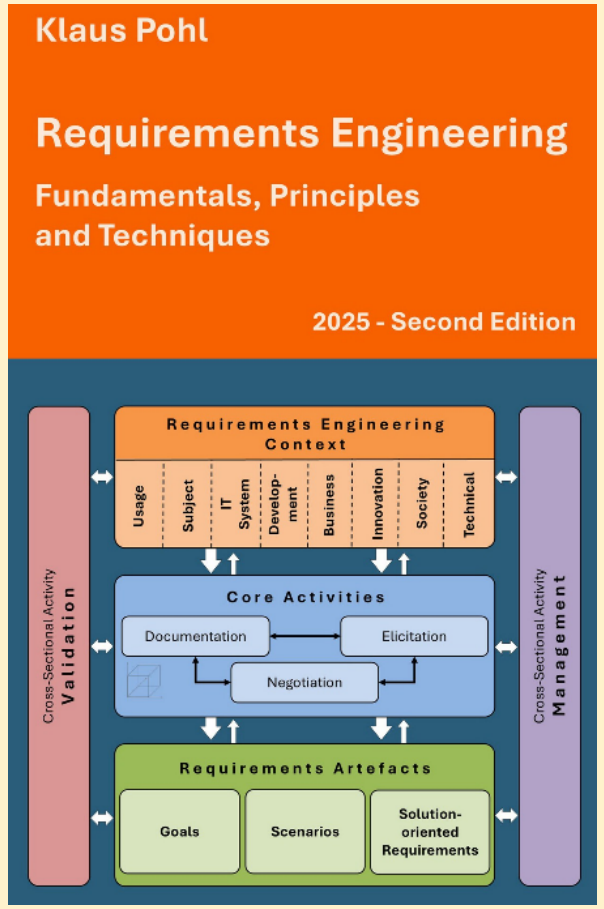
- Rehearse in-class demonstrations – *more than once*

- Some students won't use AI
  - Typical practice problem set: 6% no attempt; 11% "garbage" answers; 10% answer all but AI questions
    - Some students explicitly said they refused to use AI

- Intro course: can still teach CS1 concepts and also incorporate GenAI
  - But I need to find room to make it more than a bolt-on topic

CSCE/SOFT 468/868

- Requirements Elicitation, Modeling, and Analysis – Spring 26
- Juniors and Seniors CS and SE majors – required course
- Offered every other year in Spring
- A lot of in-class group work.



Assignments 30%
Readings and Reflections 5%
Class Participation 5%
Midterm 20%
Group Project 40% - 3 milestones

The hardest part of building software is deciding precisely what to build – Fred Brooks

Pilot - Re-designing the course with industry feedback

~100 students

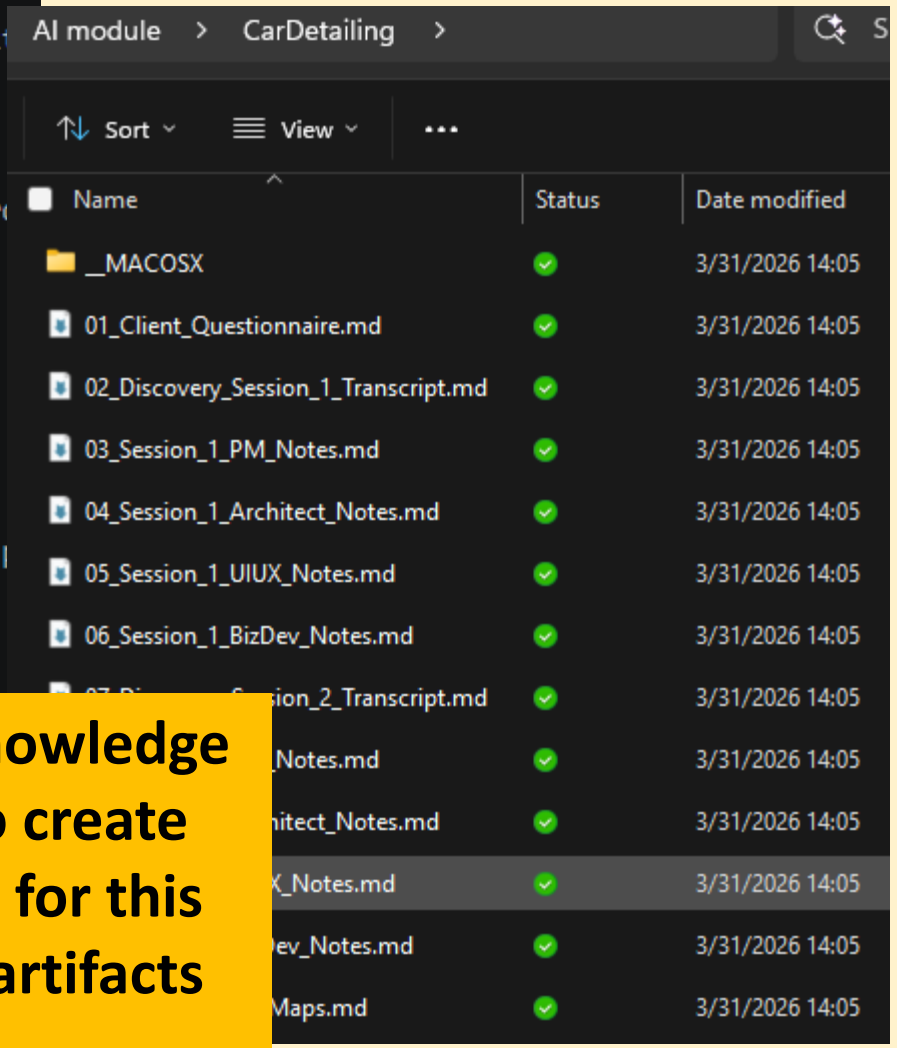
How AI was incorporated into the class

- Syllabus and assignments clearly stated when and how AI was allowed per assignment.
- During early weeks – free form prompting to create goals and requirements, and goal refinement graphs.
  - "List 100 requirements, domain assumptions, and goals." - Not fruitful.
- Mid-semester – introduced a lecture in collaboration with Don't Panic Labs on **AI-Assisted Requirements**
- Assignment 4 – AI-assisted scaffolded refactoring assignment in Python.
- **Final Group Project** – AI Usage allowed – disclosure of models and method of use were required.

AI-Assisted Requirements lecture – with Don't Panic Labs

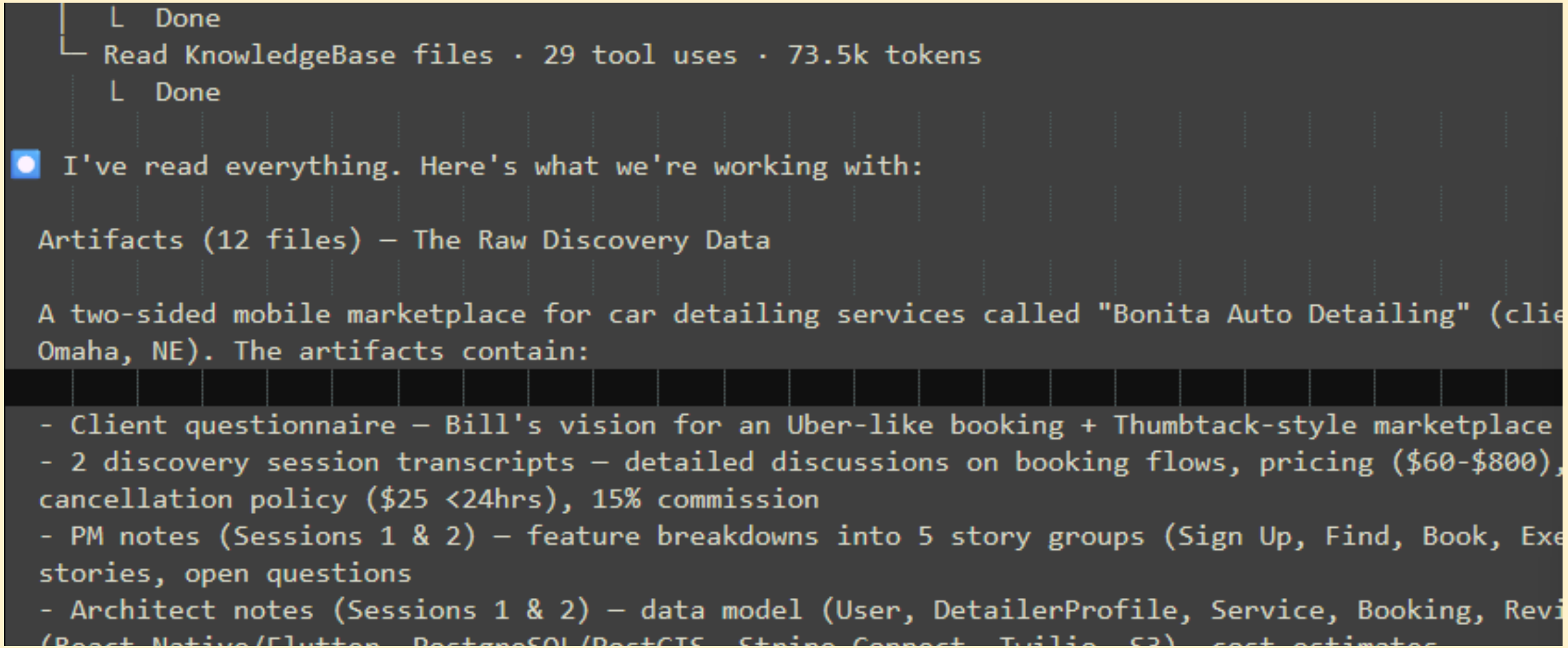
- Where do requirements live? - scattered and evolve
- Acceptance Criteria. Gherkin style. Make requirements testable
- Many markdown files -> structured requirements
- Process
  - Gather raw Inputs (meeting notes, feature requests, story maps, tickets, conversations, etc...)
  - Feed them to an AI tool with a clear prompt (Ask Mode – to explore and clarify scope)
  - AI extracts and structures user stories, functional requirements, acceptance criteria
  - Review, edit, and iterate
  - Output: Clean requirements in markdown (Plan Mode – to structure output)

```
# KnowledgeBase

## Overview

This KnowledgeBase provides comprehensive instructions for AI agents to understand and work wi

## Core Architectural Principles

- **Manager Layer**: Where most business logic happens - business logic orchestration and coor
- **Engine Layer**: Reusable, specialized business logic components for repetitive operations
- **Accessor Layer**: Data access and external service integration with resilience patterns
- **Utility Layer**: Truly universal cross-cutting concerns consumed by all layers
- **Critical Rule**: Services in the same layer can NEVER call each other

## Discovery Process Validated ✅

Our workflow has been successfully tested with Admin Manager Interface project. The discovery

## Navigation Guide

### Core Architecture

- **[Development/architecture-overvi
- **[Development/common-interface-pa
- **[Development/interface-project-p

### Development Standards

- **[Development/naming-conventions.
- **[Development/development-patterr
- **[Development/accessor-patterns.m
- **[Development/quick-reference.md](Development/quick-reference.md)** - Quick reference for AI
```



Prompt ❯ Read the contents of the knowledge base directory, we need to prepare to create requirements and acceptance criteria for this system based on the contents of the artifacts folder.

Claude Output – Car Detailing App Example



Final Group Project – Based on their capstone projects – 3 milestones

- Generate end user interview questions for different roles. Structured process.
- Consolidate end user interview transcripts.
  - Stakeholders speak in goals not specs
  - Requirements evolve and the evolution is not always documented
- Generate context diagrams and conceptual models, journey maps
- Generate functional and quality requirements, constraints, goals, domain assumptions, goal requirement graph
- Focus on security requirements
- Time logs were required per person per group
- A mandatory meeting with all groups and instructor was held in Week 12 to discuss artefacts and progress.
- **Final Deliverables** – markdown files of the requirements specs including diagrams (source + image)

Struggles with AI – Instructor Facing

- Three students refused to use any form of LLMs on their machine or in their code editor.
- Assessment challenges
  - How can an instructor assess that students have learned?
- Reflection challenges
  - Students tend to struggle with reflections in general. Reflections on the use of AI were even harder to quantify.
  - No guarantee that they used AI to do the reflection itself, which defeats the purpose.

Reflections on Current Literature Every Week

Why Johnny Can't Think: GenAI's Impacts on Cognitive Engagement

RUDRAJIT CHOUDHURI, Oregon State University, USA
CHRISTOPHER A. SANCHEZ, Oregon State University, USA
MARGARET BURNETT, Oregon State University, USA
ANITA SARMA, Oregon State University, USA

Context: Many students now use generative AI (genAI) in their coursework, yet its effects on their intellectual
poorly understood. While prior work has investigated students' cognitive offloading during episodic interaction
whether using genAI *routinely* is tied to more fundamental shifts in students' thinking habits.
Objective: To explore this possibility, we investigate (RQ1-How): how students' trust in and routine use of genAI
engagement—specifically, reflection, the need for understanding, and critical thinking in STEM coursework. Fu
(RQ2-Who): which students are particularly vulnerable to these cognitive disengagement effects.
Method: We drew on dual-process theory, cognitive offloading, and the automation bias literature to develo
explaining how and to what extent students' trust-driven routine use of genAI affected their cognitive engager
coursework, and how these effects differed across students' diverse cognitive styles. We empirically evaluated thi
Least Squares Structural Equation Modeling on survey data from **299** STEM students across five North Americs
Results: Students who trusted and routinely used genAI reported significantly lower cognitive engagement. Un
with higher technophilic motivations, risk tolerance, and computer self-efficacy—traits often celebrated in STEM-
these effects. Interestingly, students' prior experience with genAI or academia did not protect them from cognit
Implications: Our findings suggest a potential *cognitive debt cycle* in which routine genAI use progressivel
intellectual habits, potentially driving over-reliance and escalating usage. This poses critical challenges for curricu
design, requiring interventions that actively support cognitive engagement.

CCS Concepts: • **Human-centered computing** → **Empirical studies in HCI**.

Speed at the Cost of Quality

How Cursor AI Increases Short-Term Velocity and Long-Term Complexity in Open-Source Projects

Hao He
Carnegie Mellon University
Pittsburgh, Pennsylvania, USA

Courtney Miller
Carnegie Mellon University
Pittsburgh, Pennsylvania, USA

Shyam Agarwal
Carnegie Mellon University
Pittsburgh, Pennsylvania, USA

Christian Kästner
Carnegie Mellon University
Pittsburgh, Pennsylvania, USA

Bogdan Vasilescu
Carnegie Mellon University
Pittsburgh, Pennsylvania, USA

Abstract

Large language models (LLMs) have demonstrated the promise to revolutionize the field of software engineering. Among other things, LLM agents are rapidly gaining momentum in software development, with practitioners reporting a multifold increase in productivity after adoption. Yet, empirical evidence is lacking around these claims. In this paper, we estimate the *causal* effect of adopting a widely popular LLM agent assistant, namely Cursor, on *development velocity* and *software quality*. The estimation is enabled by a state-of-the-art difference-in-differences design comparing Cursor-adopting GitHub projects with a matched control group of similar GitHub projects that do not use Cursor. We find that the adoption of Cursor leads to a statistically significant, large, but transient increase in project-level development velocity, along with a substantial and persistent increase in static analysis warnings and code complexity. Further panel generalized-method-of-moments estimation reveals that increases in static analysis warnings and

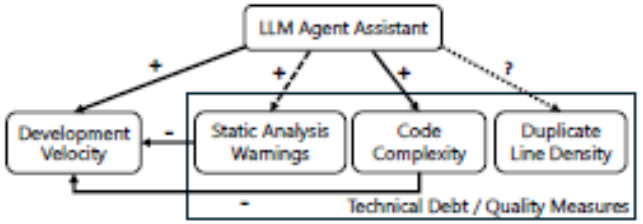


Figure 1: Our theory around how LLM agent assistants impact software development. Solid lines indicate causal relationships supported by our analysis; dashed lines indicate partial evidence; and dotted lines indicate inconclusive evidence.

with autonomous capabilities to inspect project files, execute commands, and iteratively develop code—represent a particularly promis-

Lessons Learned – Requirements Elicitation

- Add two group project graded oral check-ins with instructor to check for understanding on core concepts in the presence of AI usage
- Explain advantages of using different diagram editors mermaid.js vs. PlantUML vs. GraphViz.
- Explicitly explain model + layout = picture. Need to keep the model for editing. Reinforce this habit especially with online live editors.
- More worked-out examples using AI to see when it fails and how to spot failures. Some of this comes with experience.
- The use of AI early on in requirements elicitation can help with intent and finding gaps in the requirements/interviews before design and coding starts. Reviewing of what is presented is key.

Students' suggestion – Make space conducive to in-class group work

This image is AI-generated using CoPilot.

Classic Problem

Unclear requirements are one of the biggest source of reworking existing features.

What did the client ask for vs. **What did the engineer build** vs. **What did the client actually need**

Acknowledgements

We extend our sincere thanks to the product designers, software architects, and engineers at Don’t Panic Labs for their collaboration and invaluable contributions to this pilot course redesign.

- Bill Udell
- Anne Ruskamp
- Bob Whitmer
- Chad Michel
- Jeff Kodad

AI-Assisted Refactoring Assignment 4

- AI-assisted scaffolded refactoring assignment in Python within VS Code using AI model of choice. CoPilot was the default.
- Justification for (not) incorporating AI output + All Logs submitted
- Deliverables included updated design, before/after UML class diagrams, and reflections on the use of AI

What AI Does Well

- Extracting implicit requirements from unstructured narratives
- Finding inconsistencies and gaps across multiple sources
- Generating boilerplate structure for user stories and acceptance critieria quickly
- Asking clarifying questions you might miss - "What edge cases am I missing for this feature?"

What Software Engineers are STILL Responsible for

- Stakeholder alignment and sign-off on scope and contracts
- Prioritization and trade-off decisions
- Domain knowledge that the AI has no idea about
- Catching hallucinations before they become features