



Software Development with Agentic Coding Tools

Modern Software Development

Zachary G. Ives & Harry Smith
University of Pennsylvania
Fall 2026

Why Are We All Here Today?

- To learn about automated coding, aka "vibe coding"?
- To learn about building AI agents?

Not to set ourselves up to disappoint you, but this course is focused on something else!

A question that predates AI -- how do we go from code to **production** software?

- What makes a "10x engineer"?
- How do agentic tools change the landscape?
- And yes, how does one build agentic tools...

Software Is about *Engineering*, Not Coding!

Software engineering has *always* been about more than writing code! It is about **designing systems, managing complexity**, and **ensuring reliability**.

- Senior engineers focus on **architecture, review, and judgment**.
- Junior engineers generally *implement*.

With AI coding agents:

Human expertise shifts from **coding** to **understanding, specifying, reviewing, and verifying**.



Understand at multiple levels of detail:
leaf, branch, tree, *and* forest!

Who Are *We*?

- Harry Smith (call me "Harry")
- Contact: Ed for most things;
`sharry@engineering.upenn.edu` for private things
- Office Hours: Wednesdays 2:30-3:30 in Levine 260 (tentatively)



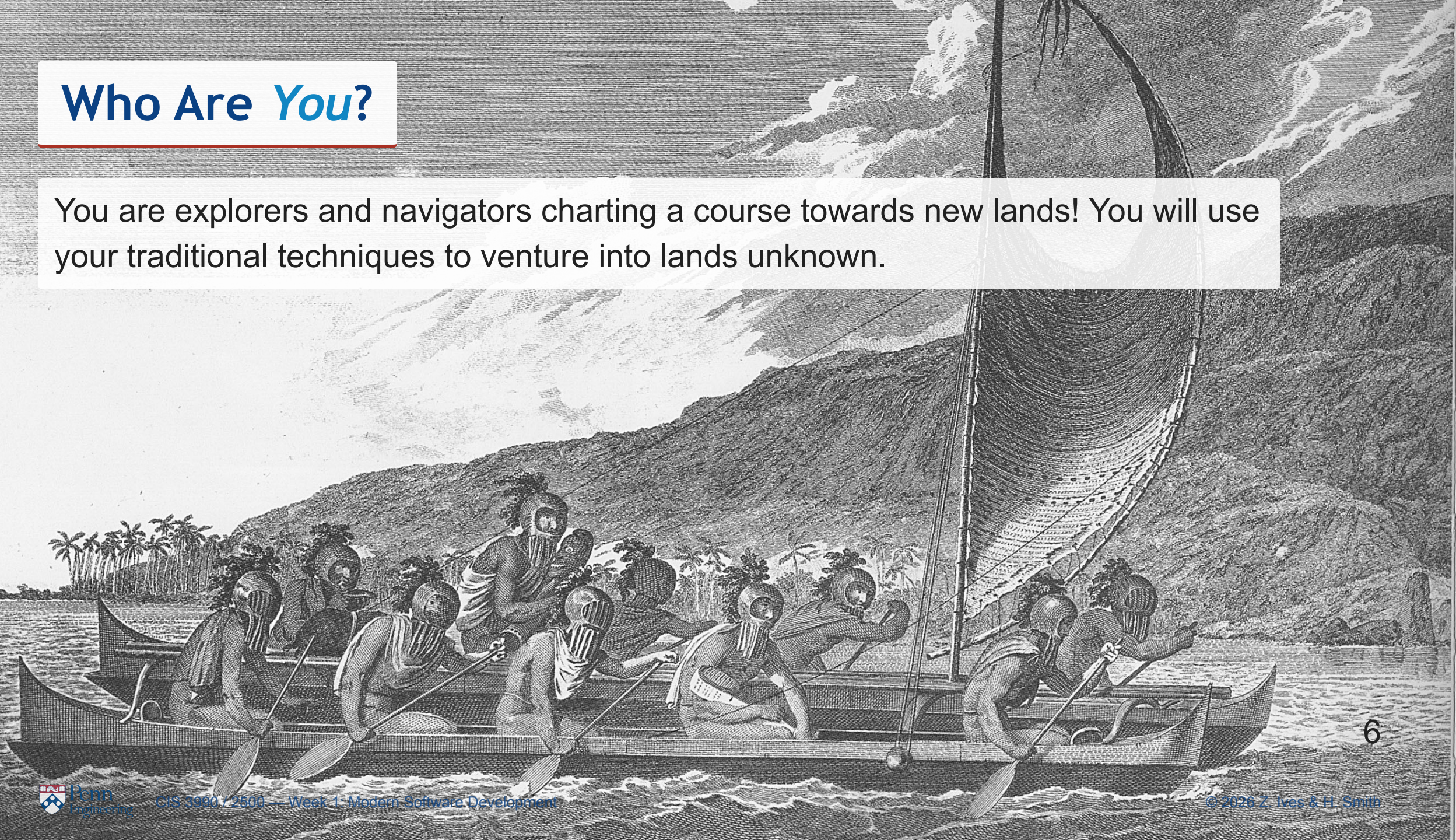
Who Are *We*?

- Zack Ives
- Contact: Ed for most things;
`zives@engineering.upenn.edu` for private things
- Office Hours: Tuesdays 4:30 in Levine 305
(starting in 2 weeks), or send me an email to find a time
- Research areas: AI systems, data management, AI for science
- Agentic systems developed: [AIRFoundry KAIR platform](#)
(AI for RNA therapeutics), [Agent P](#) (platform for Penn agentic courses!)



Who Are *You*?

You are explorers and navigators charting a course towards new lands! You will use your traditional techniques to venture into lands unknown.



And Yes, We'll Be Using Agents!

You've built small projects. This course will work on a **big open-source project**.

- A clone of the Discord chat server called Spacebar: a **40,000-line, 780-file** codebase

Next time we'll use **Claude Code** and Penn's **PARCC** to start to understand it!

- *"Trace a chat message from the client's send button to another user's screen."*
- *"Draw me a diagram of the main modules"*

Your goal should be to use agents to their fullest, not just to have them write code!

An agent that can explore a codebase faster than any human — but also invent function names with confidence — needs an engineer who knows enough to tell the difference.

You Know about the Speed of the AI Coding Revolution

YEAR	OBSERVATION
2022	GitHub Copilot: ~40% of new code at some orgs AI-assisted
2024	GitHub Octoverse: AI-generated code dominant at leading companies
2026	Anthropic: 40–100% of production code at top companies written with agents

Industry no longer needs novice engineers to write boilerplate code to someone else's specs.

Design, attention, specification, and verification are now the scarce engineering resources.

It needs engineers who can make big-picture judgments and **own systems they did not write.**

Engineering in 2026

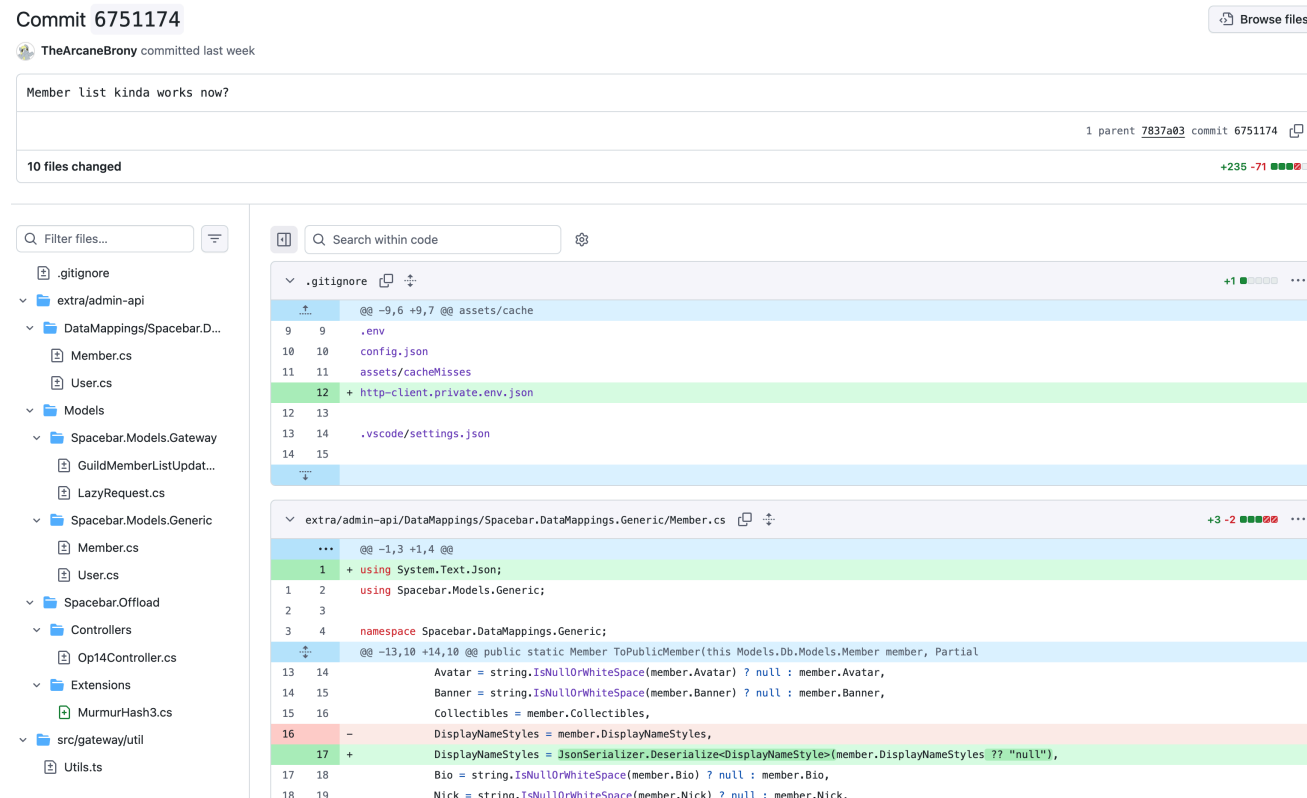
What the job actually looks like at companies hiring Penn graduates:

- Most new production code is initially drafted by an agent — not typed from scratch
- Engineers spend more time specifying, reviewing, debugging, and testing AI-generated code than writing their own
- The person who ships the most reliable software is the best reviewer — not the fastest typist

The scarce skill is good judgment and an spirit of inquiry.

What 2026 Software Engineering Actually Looks Like

It is your first week as an intern. Your tech lead assigns you to review a **pull request**—a proposed set of code changes that teammates must approve before they merge into the shared codebase.



The screenshot displays a GitHub pull request for commit 6751174. The commit message is "Member list kinda works now?". The pull request shows 10 files changed with a net change of +235 lines and -71 lines. The file explorer on the left lists the project structure, including folders like "extra/admin-api", "DataMappings/Spacebar.D...", "Models", "Spacebar.Models.Gateway", "Spacebar.Models.Generic", "Spacebar.Offload", "Controllers", "Extensions", and "src/gateway/util". The main area shows the code diff for the file "extra/admin-api/DataMappings/Spacebar.DataMappings.Generic/Member.cs". The diff highlights changes in the "ToPublicMember" method, specifically adding a line for "DisplayNameStyles" (line 17) and modifying the "Banner" property (line 14).

Commit 6751174

TheArcaneBrony committed last week

Member list kinda works now?

1 parent 7837a03 commit 6751174

10 files changed +235 -71

Filter files... Search within code

.gitignore

extra/admin-api

DataMappings/Spacebar.D...

Member.cs

User.cs

Models

Spacebar.Models.Gateway

GuildMemberListUpdat...

LazyRequest.cs

Spacebar.Models.Generic

Member.cs

User.cs

Spacebar.Offload

Controllers

Op14Controller.cs

Extensions

MurmurHash3.cs

src/gateway/util

Utils.ts

@@ -9,6 +9,7 @@ assets/cache

9 9 .env

10 10 config.json

11 11 assets/cacheMisses

12 + http-client.private.env.json

12 13

13 14 .vscode/settings.json

14 15

extra/admin-api/DataMappings/Spacebar.DataMappings.Generic/Member.cs

@@ -1,3 +1,4 @@

1 + using System.Text.Json;

2 using Spacebar.Models.Generic;

3 namespace Spacebar.DataMappings.Generic;

@@ -13,10 +14,10 @@ public static Member ToPublicMember(this Models.Db.Models.Member member, Partial

13 14 Avatar = string.IsNullOrEmpty(member.Avatar) ? null : member.Avatar,

14 15 Banner = string.IsNullOrEmpty(member.Banner) ? null : member.Banner,

15 16 Collectibles = member.Collectibles,

16 - DisplayNameStyles = member.DisplayNameStyles,

17 + DisplayNameStyles = JsonSerializer.Deserialize<DisplayNameStyle>(member.DisplayNameStyles ?? "null"),

17 18 Bio = string.IsNullOrEmpty(member.Bio) ? null : member.Bio,

18 19 Nick = string.IsNullOrEmpty(member.Nick) ? null : member.Nick,

What 2026 Software Engineering Actually Looks Like

You have two choices:

- Click *"Approve"* — your coworkers are smart and you can't understand all of it before the deadline anyways. 🙄
- Understand the changes and reason about their impact; that is, **review!**

The second choice is what you will learn to do in this course:

- read PRs
- understand the execution path of the proposed change
- predict failure modes
- write tests to catch those failures.

Summary: "Coding" vs. Software Engineering

Coding (~CIS 1200)

- Generating text that compiles as code
- Satisfying a local requirement
- Thinking in lines
- Measured in output

Agents do this for "free". But not always reliably.

Software Engineering

- Designing systems that remain correct at scale
- Managing complexity, interfaces, and invariants
- Thinking in architectures
- Decomposing into components that can be worked on independently
- Measured in reliability and maintainability

Agents need a human to own this.

You Must Do Your Homework

Even when AI writes code, you must **own** that code. Ownership requires intuitive understanding.

Sometimes this means **going through the motions ourselves!** In the classic *Karate Kid*, Daniel learned muscle memory by doing repetitive chores.

This course works the same way:

- Reading type annotations carefully now → pattern recognition that catches silent errors
- Doing permission-bit arithmetic by hand now → intuition that spots the subtle security bug in seconds



The Engineer as Team Manager

You are not here to write code. You are here to...

- **Direct** — write precise specifications an agent can act on
- **Review** — understand code you did not personally type; catch what agents miss
- **Verify** — tests, continuous integration (CI), and behavioral contracts are your accountability layer
- **Iterate** — architecture, debt, and security are designed in, not bolted on later

This course builds those four capabilities across 14 weeks.

Durable skills: Practice specification-writing, mental-model building, and adversarial testing — so your abilities don't atrophy when agents do the typing.

Risks and Three Failure Modes of Agentic Development

AI-assisted software at its worst results in buggy software that no developer understands, and that is only fixed by constantly asking AI to fix an observed issue.

Each failure type is the predictable result of skipping a specific discipline.
The course provides the "antidotes."

- 1. Shallow mental models** — accepting agent-written code without understanding it
- 2. Accelerated technical debt** — agents optimize locally, compound debt exponentially
- 3. Inadequate review, testing, and security** — plausible code hides logic errors

Failure Mode 1: Shallow Mental Models

You accept agent-written code without understanding it.

The codebase becomes a black box you cannot debug or extend.

Example: Your agent calls Spacebar's function to `checkUserPermission` on a channel. You don't study it, so when a user sees unauthorized channels, **you can't locate the bug because you never built a mental model of how permissions resolve.**

Antidotes in this course:

- Deliberate code comprehension
- Creating and updating a Developer Guide
- Diagrams of control flow and modules

Failure Mode 2: Accelerated Technical Debt

You let the AI keep tacking on new features. The lack of a systematic architecture results in duplication and irregular interfaces -- **technical debt** you will need to pay back later, possibly in debugging!

Agents optimize locally and generate debt *exponentially faster* than human teams. Each new prompt regenerates code from context rather than reusing existing abstractions.

Example: Your agent adds a new permission check function each time you do something new -- each doing a slightly different thing. What does what? Do they work?

Antidotes in this course:

- Architecture-first thinking
- Give the agent guidelines, e.g., **CLAUDE.md** and Architectural Decision Records

Failure Mode 3: Inadequate Review, Testing, Security

Your agent generates plausible code that hides logic errors and vulnerabilities.

Compiling the project doesn't catch these.

Example: The agent generates user passwords using a cryptographic library -- but that has been broken since 1996. It repeats the mistake LinkedIn made, which lost 6.5 million password hashes to a hacker in 2012!

Antidotes in this course:

- Specification-driven testing
- Continuous integration prevents commits until "green"
- Security and adversarial robustness as first-class concerns

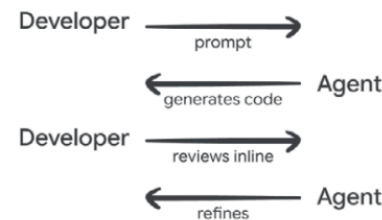
What This Means for You

A recent whitepaper from Google summarizes the current "Software Development Lifecycle." It posits that an engineer now alternates between two roles.

What do you notice about these two roles?

[\(summary blog with link to paper here\)](#)

Conductor Real-time • Synchronous • In-IDE



Inline Completion

Chat-in-Editor

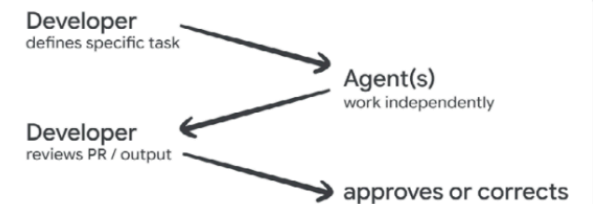
Diff Review & Accept/Review

Quick Fix / Refactor

- Keystroke-level control
- Immediate feedback
- Single-file scope
- Developer always in the loop

Best for: Exploratory coding, prototyping, learning a new API

Orchestrator Asynchronous • High-level • Multi-agent



Task / Issue Assignment

Background Terminal Agents

CI/CD Integration

Eval & Test Suites

Multi-Agent Coordination

- Goal-level control
- Delayed feedback
- Multi-file scope
- Reviews outcomes not keystrokes

Best for: Feature implementation, migrations, test generation

There's a Vital Spot for YOU 🧠 in This Process

As an engineer working with AI agents:

- You 🧠 **read and understand** code before merging it
- You 🧠 **write tests** that define correct behavior, independent of the implementation
- You 🧠 **build CI** that checks behavior automatically on every change
- You 🧠 treat the agent as a skilled but fallible subordinate — not an authority

And if you don't...

🧠🧠🧠🧠🧠 YOU 🧠🧠🧠🧠🧠 are responsible!

So -- What Are We Focused on?

This course is about **software engineering as design**: decomposition, clean interfaces, specification, review — the things that decide whether a system stays correct as it grows, regardless of which tool happens to be trendy this year.

And: the questions and concerns you should be focusing on, regardless of whether humans or agents are at work!

You will need to **engage very actively with the AI** - with critical inquiry, not just taking its responses at face value or asking the obvious things!

Course Prerequisites

Required: CIS 1200 (Intro Programming — OCaml & Java) and CIS 1210 (Data Structures & Algorithms — Java).

We assume you know

- Java, comfortably
- Basic algorithms and data structures
- Some OCaml from CIS 1200

You'll teach yourself (using LLM assistance)

- How to read a 40K line code base!
- JavaScript / TypeScript
- Git or the command line
- Databases, test-driven development

How the Semester Breaks Down

Roughly, the topics

1. The engineer's new role (*Wk 1*)
2. Managing agentic tools (*Wk 2–3*)
3. Building good software with AI (*Wk 4–5*)
4. Specifying & validating needs (*Wk 6–8*)
5. Iteration & validation tooling (*Wk 9–11*)
6. Performance, security, privacy (*Wk 12–14*)

Roughly: the grading scheme

- Weekly Labs -- use what we talk about (**15%**)
- 3 sizable Projects, including a capstone, that build on the labs (**65%**)
 - A significant portion of the grade comes from **presentations**
- Midterm (**15%**)
- Attendance & Demos (**5%**)

Demos

In-class activities that will encourage you to explore and evaluate a codebase live. We'll use `agent-p.org`, a webapp designed for this course.

Let's try one!

Labs

A lab sits somewhere between a *problem set* and a *project milestone*. They are:

- something to **do** with the material for each week
- graded Pass/Fail for completion & reasonable correctness
 - (this is not a course where we can always proclaim in advance what "correct" would be!)
- mostly 1-3 hours-long activities—not supposed to be especially onerous!
- useful conversation starters with us, TAs, each other on Ed
- **vital steps for completing the larger homework projects**
 - although our first project won't release for a few weeks, the labs that precede it form essential scaffolding for actually completing it.

Lab 1 at a Glance

Assigned: Tuesday Aug 25 · **Due:** before Tuesday Sep 1

Parts A-C: Environment setup

- Install Claude Code; point at LiteLLM proxy
- Clone Spacebar; install dependencies (`npm install`)
- Run the server locally; verify `curl http://localhost:3001/healthz` → `200 OK`
- **This could take a little while. Please start early.**

Part D: Git basics

- Create a feature branch, make a small change, open a PR against `main`
- Understand the commit → push → PR → review loop

Part E: CI bootstrap

- Create `.github/workflows/ci.yml` using the workflow from Thursday's lecture
- Confirm the green checkmark appears on your PR

26

Labs Projects

Before the first project, we complete four labs.

LAB #	TOPIC
1	Environment Setup
2	Getting up to Speed in TypeScript
3	Understanding a New Codebase (for You and an Agent)
4	Building a Formal Record of a Project's Architecture

The first project, tentatively released on **Sep 17**, requires robust comprehension that will come from completing these labs.

Projects

Projects are larger and more open-ended than labs, requiring you to improve or extend the codebase in a meaningful way.

- the first project will have you use tests to enforce specifications in systems that you didn't write
- the second project allows you to work in a team to implement (and test) a new feature
- the capstone project has a few different tracks, allowing you to choose between:
 - MCP integration
 - Technical debt cleanup
 - (you'll get practice with both from Labs regardless of your choice of track.)

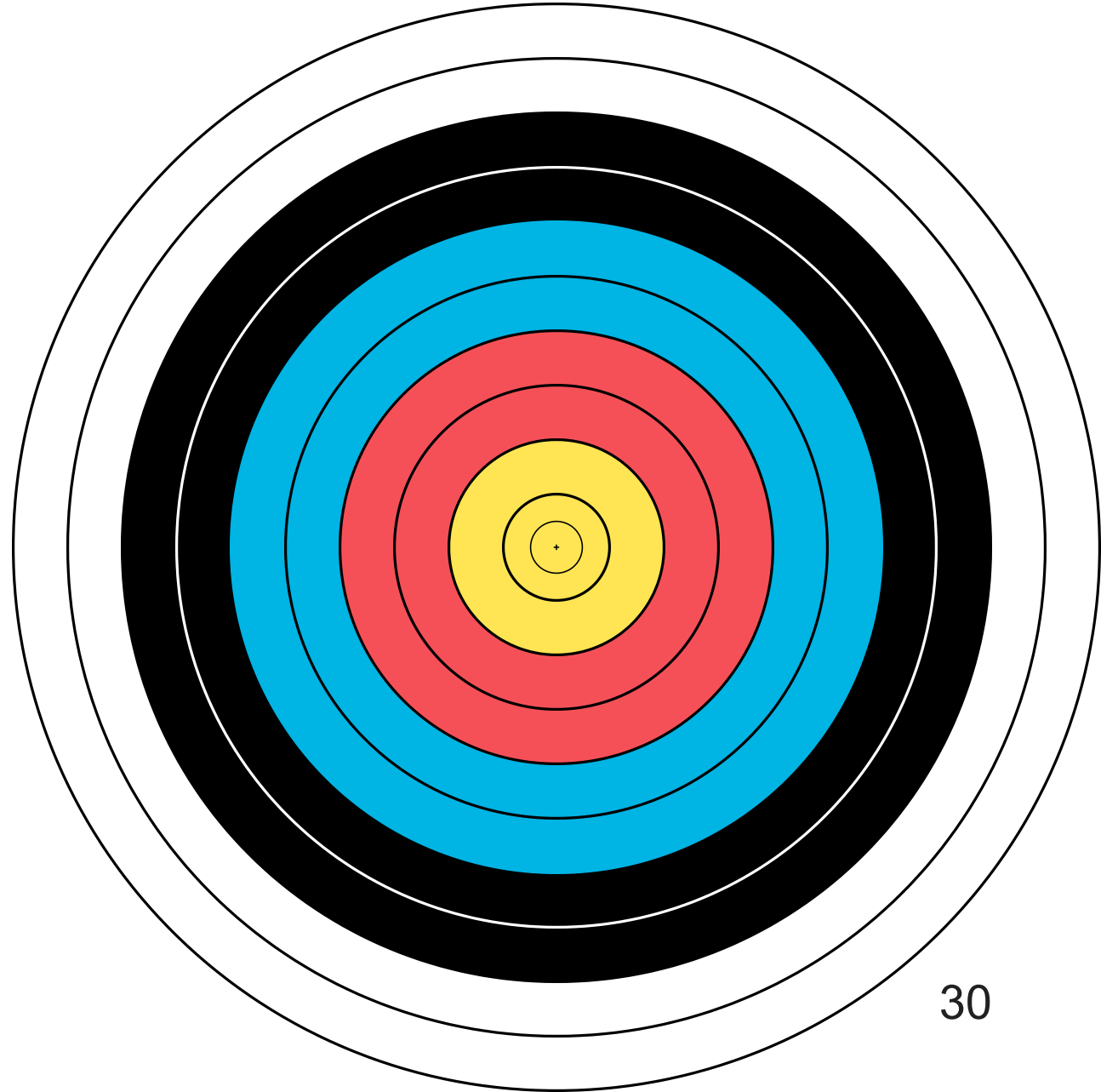
Midterm

More on this as we approach the date :)

Projects & Workload

It's foolish for an instructor to try to how "easy" or "hard" a course will be...

- there are multiple dimensions to difficulty: complexity, workload, etc.
- different students respond differently
- this is the first time this course is offered (here; anywhere, really)
- we'll ask you to *do a lot* but the point of the course is to give you tools to *do more with less!*



A Disclaimer: This Is a Bleeding-Edge Course

This is a pilot of a course that will join the CIS core. *You are the early adopters!*

- Caveat emptor.

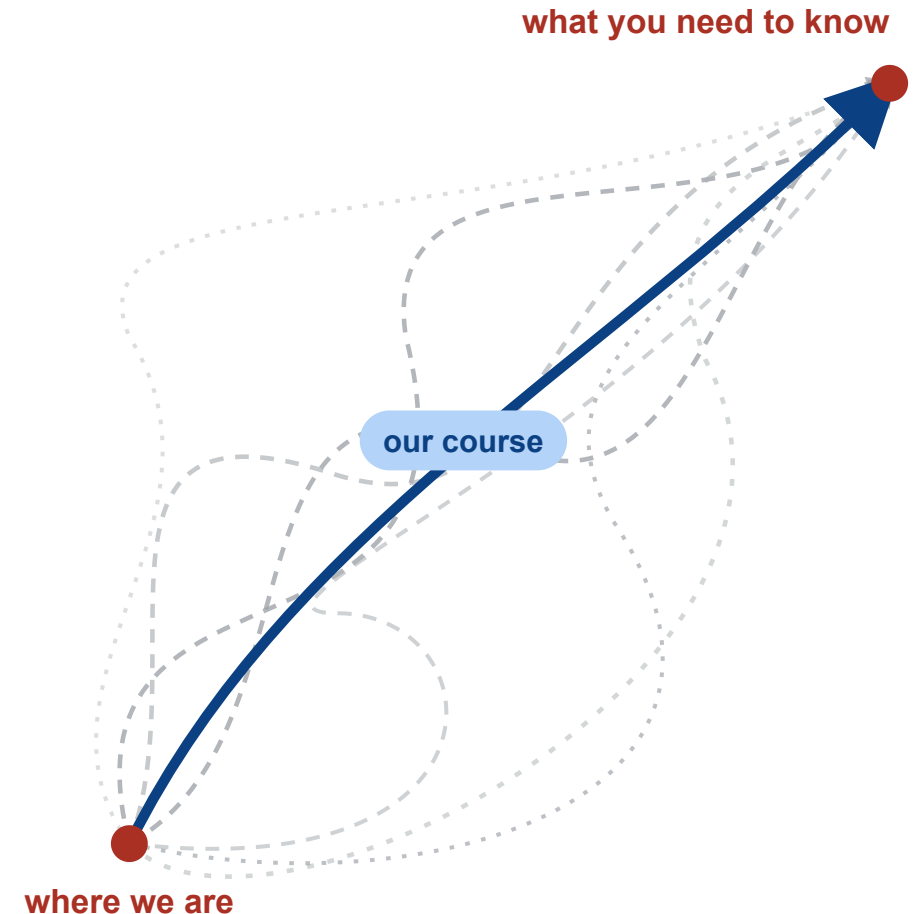
The tools and practices are changing *while the semester runs*. What's true in September may shift by December.

- We will be using Claude Code with an open model hosted on Penn's PARCC, which may have its bumps.
- Materials, labs, and this syllabus will be revised as we learn what works.

Life on the Bleeding Edge: A Philosophy

LLMs have created exciting upheaval in the world of... everything! We, altogether, are charting a course into an uncertain future.

- As instructors, we have expertise that we'll use to guide the course.
- As students, you're learning the field as it establishes itself.



Life on the Bleeding Edge: A Philosophy

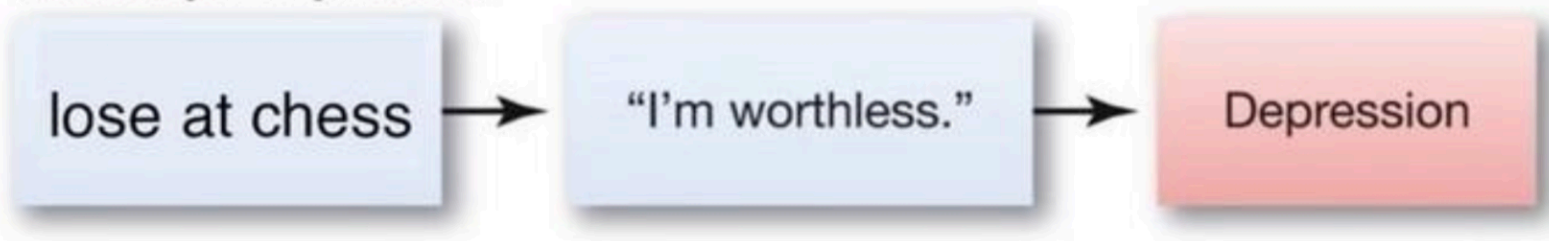
We should not pretend that what we are doing with AI is always durable and “right” in a long term sense. The course's assignments are designed to be open-ended and include experimentation. We will learn from you as you learn from us.

Life on the Bleeding Edge: A Technique

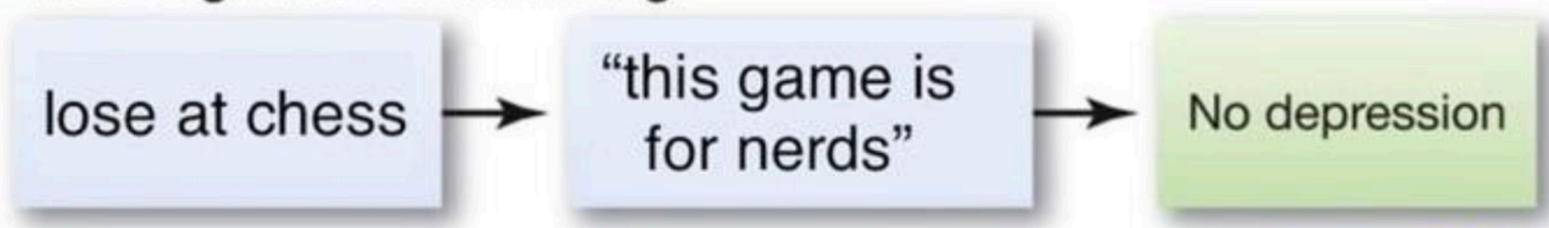
(this and the next two slides are about 50% a joke)

Cognitive restructuring is a technique in cognitive behavioral therapy that allows you to recontextualize your thoughts in order to avoid harmful patterns in thinking...

Maladaptive pattern:



After cognitive restructuring:



Maladaptive pattern:

I spent a lot of effort
working through a
complex project.



"That was so hard
and I don't even know
if I was right or not."



Disappointment

After cognitive restructuring:

I spent a lot of time
working through a
complex project.



"I had to think hard and
made imperfect choices,
but next time I'll be on a
better path."



Growth

Life on the Bleeding Edge: The Rewards

By participating in this course with all of its experimentation and discovery, you'll also be more prepared for the modern software engineering landscape than virtually anyone doing a Bachelor's!

So:

1. please be patient with some inevitable glitches,
2. please give us feedback on how to make the course even better!
3. if you have low tolerance for this, keep in mind that this course will be offered again in an updated version in the Spring!

Recap: The Engineer's New Role

- **Coding and software engineering are different activities** — agents can do the former for free; the latter still needs a human who understands architecture, complexity, and invariants.
- **Three failure modes drive the whole course arc**: shallow mental models, accelerated technical debt, and inadequate review — each is what happens when a discipline gets skipped.
- **The scarce skill is judgment, not typing speed** — direct, review, verify, iterate.

Questions?

`zives@engineering.upenn.edu` · `sharry@engineering.upenn.edu`