# Working with coding agents

CSE-416-03 · Sep 9, 2026

# Recap

- M1 is next week. You need a written proposal in the repo. Put the repo/doc link into the team spreadsheet.
- Received many questions.

# Q&A

- Q: We brainstormed these features, can you see if they are good?
- A: I'm not your user, but I don't need to be! You would want to find some evidence to support your features, and show that your brainstorming have real users. That's the idea. You don't need my approval to do idea. What I will "Judge" is whether you have spend reasonable amount of efforts to evaluate your idea if they have real users potentially.

# How do we expect from your proposal?

- Is there a proposal?
- Is this proposal carefully generated?
- Are there efforts spent on studying users’ needs?
- Are the team working together?

# Heads-up on the presentation

- Come 10 minutes before class to test you laptop.
- Try not to switch computers during your presentation.
- You don’t have to all speak.

# Today

- Today is how to use a coding agent on that work — and for the rest of the semester.
- Some of this is from Stanford CS146S (**The Modern Software Developer**).
- Three parts: **prompt engineering**, **context engineering**, and **specification engineering**.

# Autocomplete vs an agent

# Autocomplete, chat, agent

### Autocomplete

Completes the line. Useful. You are still writing the program.

### Chat

You ask a question, copy an answer, run it yourself.

### Agent

It edits files, runs commands, reads the error, tries again. You stop it.

# Three ways to optimize

### Prompt

Unclear request. Rewrite what you ask for this turn.

### Context

Missing evidence. Add the right files, output, or tools.

### Specification

Work spans turns or people. Write the agreement down.

- These overlap in real work. None of the three is “the most important” for every task.

# Prompt engineering

# Prompt engineering

- You are choosing the words in the box so this turn does the right thing.
- Still worth doing. “Add tests” and “add tests for a $0 bill and a roommate who already paid” are not the same request.
- It cannot fix missing files or a messy repository. Those are different problems.

# Say the cases

### Weak

“Add unit tests.”

For what? It will test the happy path, or invent cases you do not care about.

### Better

“Add tests for: amount 0, a negative amount, and a roommate who already paid. Do not add a chat feature.”

Same idea if the work spans DB + API + UI: one piece per prompt, then look at it.

# Show the shape you want

### Weak

“Write a good commit message for this diff.”

“Good” is not a format. You get a paragraph, or something you will rewrite.

### Better

“Same style as these:”

```
Fix split so a $0 bill does not crash

Mark a bill paid from the list, not the detail page
```

Then: “One line. What changed, not how you felt about it.”

- One or two examples in the prompt beat a long speech about quality.

# Simon Willison (Django creator): one short prompt

- Simon has a `blog-to-newsletter` tool. In 2026 he wanted it to include a new kind of post called a “beat.”
- The repository had more than 200 little HTML tools. His prompt named the one file to change.
- It also named an existing implementation to copy and told the agent exactly how to check the result.
- He reports that the resulting pull request made the right change in one pass.

Source: Simon Willison, “Adding a new content type to my blog-to-newsletter tool,” Apr 18, 2026

# The prompt

### Reference

“Clone `simonw/simonwillisonblog` to `/tmp` for reference.”

### Change

“Update `blog-to-newsletter.html` to include beats that have descriptions — similar to how the Atom everything feed works.”

### Check

“Run it with `python -m http.server`; use Rodney to test it; compare it with the live homepage.”

- It says **where to look**, **what to change**, and **how to tell if it worked**.
- Willison’s point: an existing working example often explains the desired behavior better than another paragraph.

# Ask for a plan, then stop

- “Read `spec.md` and `app/bills.py`. Tell me how mark-paid would work. Do not edit files.”
- Then you fix the plan (wrong table, extra feature, it wanted a new auth).
- Then: “Do step 2 only.”
- If you skip this, it often adds a second copy of something you already have.

# What usually does not help

- “You are a world-class senior engineer…” — it does not become one.
- “Make it production ready / perfect / secure.” No check, so it will claim it did.
- Restating the whole product in every message. That just fills the window.
- A useful extra line is a **constraint**: “Do not touch auth.” “No new dependencies.”

# Context engineering

# Context engineering

- You are choosing what the model is allowed to see this turn — not just how you phrase the ask.
- That includes: files you @-mention, `AGENTS.md`, the conversation so far, `npm test` output, a screenshot of the broken page.
- Two teammates can type the same sentence and get different code, because one attached `bills.py` and the other attached the whole `src/`.
- Most “the model is dumb today” problems are the wrong files, a stale thread, or two docs that disagree.

# Same prompt, different files

### Prompt only

“Add mark-paid.”

Empty chat, no files.

It invents a new table, a new route, maybe a points system — because it cannot see yours.

### Prompt + the right files

Same sentence, plus `spec.md` and `app/bills.py`, plus “do not change auth.”

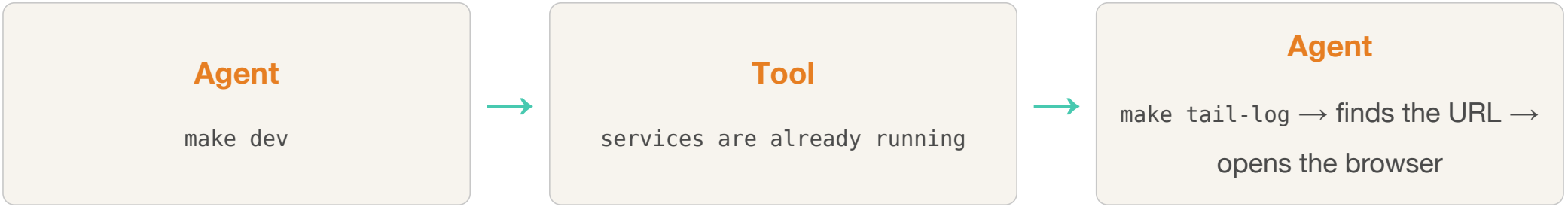
It has a chance of editing the function you already have.

# Armin Ronacher changed the environment

- Armin Ronacher (creator of Flask) uses coding agents heavily. He found that the agent would sometimes start a second development server.
- He changed `make dev` to detect that the server was already running and print a clear error.
- He also logs the server output to a file and provides `make tail-log`.
- Now the agent can see the state of the program and recover without Armin copying errors into chat.

Source: Armin Ronacher, “Agentic Coding Recommendations,” Jun 12, 2025

# What one run looked like




- For login tests, the debug server prints the email link in the log. `CLAUDE.md` tells the agent where it is.
- Context engineering here is not “attach more prose.” It is making the running system observable.

# A screenshot during an outage

- Anthropic's Data Infrastructure team had Kubernetes clusters that stopped scheduling pods.
- They gave Claude Code screenshots of their dashboards.
- It walked them through the Google Cloud UI, identified exhausted pod IP addresses, and gave commands to add an IP pool.
- Anthropic says this saved about 20 minutes. Treat the number as a vendor's own case study, not an independent measurement.

Source: Anthropic, "How Anthropic teams use Claude Code," Jul 24, 2025

# The error is the useful context

- Run it. Paste the traceback (or a screenshot). You do not need to rephrase the product.
- “It doesn’t work” gives it nothing. The exception, the failing test name, the URL — those do.
- For UI, a picture of the page beats another paragraph of “the button is on the left.”

# Specification engineering

# Specification engineering

- A prompt is a request for this turn. Context is what the agent can inspect.
- A specification is a durable description of behavior, constraints, and how you will check the result.
- It could be an issue, `SPEC.md`, an API example, a diagram, or a set of acceptance tests.

# When writing a spec helps

### A small experiment

“Can this library read the file?” Try it and throw it away.

A detailed spec would slow you down without helping.

### A feature the team will keep

Several screens, stored data, error cases, two people editing it.

Write enough that the next person does not have to recover the decisions from chat.

# Armin Ronacher: the tests were the spec

- In January 2026, Armin Ronacher ported MiniJinja from Rust to Go with an agent doing almost all of the coding.
- MiniJinja already had snapshot tests: inputs plus the exact outputs expected from the Rust implementation.
- He and the agent agreed to reuse those snapshots and port in stages: lexer → parser → runtime.
- The run lasted 10 hours; Ronacher reports about 45 minutes of active human time, 34 prompts, and $60 in API cost.

Source: Armin Ronacher, “Porting MiniJinja to Go With an Agent,” Jan 14, 2026

# Where the tests were not enough

- The agent gave up on “must fail” tests because Go could not reproduce the Rust error text exactly. Ronacher made it use fuzzy matching instead.
- It also tried to change HTML escaping and iterator behavior to make the port easier. He rejected those changes.
- Once the behavior was pinned down, the agent ran unattended for seven hours to finish the long tail.
- The lesson: executable examples are a strong specification, but a human still decides which differences are acceptable.

# Spec-driven development (SDD)

- Write and review the expected behavior, constraints, and acceptance criteria before asking an agent to implement.
- Turn the specification into a plan and small tasks; implement and check each task against it.
- Keep the specification in the repo and update it when the team changes a decision.
- Most popular tools: Spec Kit, Kiro, OpenSpec

# OrangeLoops: two hours to generate, four days to finish

- In early 2026, OrangeLoops used Spec Kit and Claude Code to build a NestJS backend and an Expo mobile frontend from project documents and base templates.
- The tools produced 195 tasks and about two hours of recorded code generation.
- Applying the visual design took about four hours. Testing, integration debugging, database seeding, review, and stabilization took four days.

Source: OrangeLoops, “Spec Kit + Claude Code Case Study,” May 2026

# OpenSpec example: a bill was paid

### 1. Propose

Describe the change: let a roommate mark a bill paid.

OpenSpec drafts the specification, design, and tasks.

# OpenSpec example: a bill was paid

### 1. Propose

Describe the change: let a roommate mark a bill paid.

OpenSpec drafts the specification, design, and tasks.

→

### 2. Review

The team fixes the draft: whole payments only, participants only, and balances must update.

# OpenSpec example: a bill was paid

### 1. Propose

Describe the change: let a roommate mark a bill paid.

OpenSpec drafts the specification, design, and tasks.

### 2. Review

The team fixes the draft: whole payments only, participants only, and balances must update.

### 3. Apply

The agent adds the data field, route, button, and tests. The team checks the tests and the UI.

# OpenSpec example: a bill was paid

### 1. Propose

Describe the change: let a roommate mark a bill paid. OpenSpec drafts the specification, design, and tasks.

### 2. Review

The team fixes the draft: whole payments only, participants only, and balances must update.

### 3. Apply

The agent adds the data field, route, button, and tests. The team checks the tests and the UI.

### 4. Archive

Move the shipped behavior into the current bill specification. Keep the completed change as history.

# Putting the three together

# If the result is bad, ask why

### Prompt problem

Wrong task. Rewrite the request; add cases and constraints.

### Context problem

Missing code or failure. Add files, logs, a screenshot, or a tool.

### Specification problem

Team disagrees or repeats an old decision. Update the shared note.

- Do not solve every failure by making the prompt longer.

# Demo

# Tools

# Next

- Sep 14–16: **Milestone 1** in class.
- About 10 minutes: what you finished, why, how the team (including AI) worked.
- Have the files in the repo. Bring a walkthrough.
- If you cannot name users, a core loop, and who owns what — you are late.