

M04 — AI in the SDLC Lab

Lab — Generative AI Boot Camp for Developers — Exported 2026-08-03

Time budget: ~20 min | **Day 2 AM** (shared with M05)

2026-08-03

Learning Objectives

Convert a user story into a complete development package using AI across the full software development lifecycle — from requirements through documentation.

2026-08-03

Concept Review: AI Across the SDLC

AI augments every phase — but doesn't replace human judgment

SDLC Phase	What AI Can Do	What Humans Must Do
Requirements	Expand user stories, draft acceptance criteria, identify edge cases from patterns	Validate against domain knowledge, prioritize based on business value
Design	Propose architecture, generate API contracts, suggest data models	Choose between trade-offs, ensure alignment with existing systems
Implementation	Generate code, refactor, complete patterns (M03)	Review every line, ensure business logic correctness
Testing	Generate test cases, identify edge cases (M07)	Add domain-specific tests, verify test quality
Documentation	Generate API docs, READMEs, code comments	Add context only humans know (why decisions were made)
Deployment	Generate CI/CD configs, IaC templates (M08)	Verify against infrastructure, test in staging

The Golden Rule

AI drafts → Human reviews → Human owns. Document AI involvement: `git commit -m "feat: add password reset (AI-assisted)"`

Traceability Checklist

- [] Commit messages note AI involvement
- [] AI-generated code reviewed by a human before merge
- [] Prompt history saved alongside significant AI-generated artifacts
- [] Business logic decisions made by humans, not AI

Exercise: User Story → Development Package

Choose a Scenario

A — Password Reset: "As a user, I want to reset my password so I can regain access to my account."

B — Activity Dashboard: "As an admin, I want to view a dashboard of user activity in the last 24 hours."

C — Your Own: Pick a user story from your actual work.

Produce These Artifacts Using AI

Work through the SDLC phases. Use at least **two different tools** — try OpenCode for generation and VS Code Chat for refinement.

1. Technical Spec (5 min) Expand the user story into implementation details. Your spec should include: endpoints, data models, edge cases, acceptance criteria. Use OpenCode:

```
opencode run "You are a senior engineer. Expand this user story into a technical specification: '[your story]'. Include: API endpoints, data model fields, edge cases, acceptance criteria. Output as a structured Markdown document."
```

2. Function Stubs (5 min) From the spec, generate skeleton code. Use **VS Code Chat** — create a new file, open Chat, and ask it to generate stubs based on your spec. Include type hints and docstrings.

3. Unit Tests (5 min) Generate tests for the happy path and 3 edge cases. Try a different tool than Steps 1–2.

```
opencode run "Read the function stubs. Generate pytest tests covering: happy path, null/empty inputs, invalid data, and boundary conditions."
```

4. API Documentation (5 min) Generate endpoint documentation from the stubs. Use VS Code Chat — select your code, ask for OpenAPI-formatted documentation.

Reflection

Where did AI help most? Where was human judgment essential? Did you catch anything in review that the AI missed?

Deliverable

Complete development package: spec + stubs + tests + docs — all AI-assisted, human-reviewed. If you used AI, your commit message should note it.

2026-08-03

Troubleshooting

Symptom	Fix
AI spec misses edge cases	Add: "Consider these edge cases: [list specific ones]"
Stubs don't match spec	Provide the spec as context: "Based on this spec: [paste]"
Tests test the wrong thing	Review each test assertion before running; ask AI to fix
All artifacts from same tool	Switch tools between steps — the cross-tool experience matters
Running out of time	Prioritize spec + stubs; tests and docs can be completed later

2026-08-03