ECE 493 Software Systems Design Project
Winter 2026
Laboratory #2
Dr. Scott Dick

Due Date: Monday Feb. 9, 2026, 11:59:59pm

Additional Resources

Discussion
The core deliverable for this class, or any software development project, is correct and working code. In this lab we look at the code generation process, using Spec-kit and Codex. Agentic development involves having the AI pass through several intermediate steps, each time reasoning about the model of code to be built, and adding those findings to its context. This added context then yields code that is of much higher quality, approaching or reaching release candidate status.

As we know, however, the AI cannot simply be allowed to run free. What has become clear in the last six months is that the AI should only build software incrementally, one small piece at a time, with the intermediate artifacts it produces each being validated before it is allowed to proceed to the next step. Our work in developing and validating detailed use cases and acceptance tests will now serve as the foundation for this incremental process; each use case will be a feature for the CMS, which will be created independently in its own branch of the source tree. As you proceed through the use cases, expect to see Codex doing a lot of rework of existing code files to accommodate each new use case. THIS IS FINE, it's the AI doing that work not you the human. Our job is to validate the high-level and detailed plans in each use case. IMPORTANT NOTE: when you see something that isn't right in any validation, don't manually fix the generated files. Re-generate them by re-issuing the previous slash command with additional instructions. This keeps the entire context synchronized.

Spec-kit introduces a multi-step workflow for each feature. These include the mandatory intermediate generations, and optional cross-checks; in our workflow, we will use all of those optional checks. These processes follows the flowchart in Figure 1:

&&&Figure 1

These commands are run within the command-line interface of your code generator. In our case, this means you install Codex, then install Spec-Kit, then set up your got repo for the project. Copy your use case and acceptance test files into that root directory, then launch Codex. The Spec-Kit commands are then available as slash commands within the Codex CLI. The command sequence in spec-kit is as follows:

1. */speckit.constitution* is run ONLY ONCE at the start of your project. This is where you define the high-level principles for your implementation. For example, I started by telling

it that all of my use cases were stored in the Markdown files UC-XX.md, and their acceptance tests were stored in UC-XX-AT.md. I also required that the CMS app be implemented in an MVC architecture using vanilla HTML, CSS and JavaScript. Codex will then prepare the repository (it can even set one up for you, but that burns tokens in Codex. Don't spend tokens doing things you can handle yourself, these cost real money.) This command should produce a file constitution.md. **Validation:** Read the constitution file, check that it repeats what you told the AI with the command. Re-generate

2. */speckit.specify* converts your specification into a format that the rest of Spec-Kit understands and follows. In the examples you will see on the web, it takes a user story and turns it into a more detailed specification, including functional requirements. This is done by reasoning about how to expand the user story. In our case, the flows from the use case should be copied directly over, and then functional requirements extracted from them.
    a. */speckit.clarify* is the first optional validation. It is intended that any ambiguities in the spec.md or constitution.md files are raised for the user to clarify at this point. Our specifications *should* have many fewer ambiguities being raised, simply because we provided so much more than user stories. However, that's not guaranteed. **Reporting:** Respond to Codex with any clarifications needed, and record the clarifications it requests and your responses for your lab report.
    b. **Validation:** After clarifications in (a), confirm that spec.md repeats the information in the use cases, especially the flows, without making any changes beyond style and grammar. Check that the functional requirements are congruent to the use case.
3. */speckit.plan* is then run to generate the high-level planning documents that will guide Codex. This is where the architecture and tech stack are fleshed out, the data model is created, and interface contracts specified.
    a. */speckit.checklist* is an optional validation that compares the generated plan and specification against common quality checklists (for UX, security, etc.) **Reporting:** For any checklist items not completed, record what they were and check that those items are necessary for the CMS. If so, then re-run the planning or specification steps as needed.
    b. **Validation:** After completing the checklists in (a), check that the architectural decisions and tech stack in plan.md match your intent from the constitution. Check that data_model.md and interfaces in the ./contracts/ directory are congruent to the functional requirements in spec.md. IN ADDITION, WHEN ALL BRANCHES ARE COMPLETE, combine all the data_model.md files into a single entity-relationship diagram.
4. */speckit.tasks* further breaks down the plan into individual task steps, which are then of an appropriate granularity for the AI to complete.
    a. */speckit.analyze* is the third optional validation. It checks spec.md, the plan files, and tasks.md for consistency, coverage gaps, and remaining ambiguities. **Reporting:** This command will result in a table being produced. This table should

be saved to a file and included in your report, along with actions taken to resolve any issues identified.

b. **Validation:** After completing any actions from (a), check that there are no blocking dependencies within the task sequence in tasks.md.

Requirements

You are to install the Codex CLI, Spec-Kit and git on your chosen computer system (Windows users, note that Codex only seems to run under the Windows Subsystem for Linux shell). Set up a git repository for your project, and copy your use cases and acceptance tests from Lab #1 into the root directory of your project. Then launch Codex, and run */speckit.init* to set up your development project.

When you have completed planning and task preparation for all branches, you will submit a report containing the constitution.md file, and all of the reporting and validation requested for each step above, zipped into a single archive. Implementation and testing of the CMS application will be examined in Lab #3.

Submission

Submit your zipped archive via the Canvas lab page by the deadline.

Grading

Constitution File	10%
/speckit.clarify report	10%
Specification validation	20%
/speckit.checklist report	10%
Plan validation	20%
/speckit.analyze report	10%
Tasks validation	20%