

# Spec-Driven Development

CS 691 / SWE 699

Fall 2025

# Logistics

- Project proposal grades up - should look at feedback in comments
- Lecture 8 reading questions due today next week at 4:30pm
- Lecture 8 activity (in class today), due by 10/24 at 4:30pm
- Lecture 9 reading questions due today next week at 4:30pm
- Project checkpoint due in 2 weeks

# Research Study

- Opportunity to share your Reflections assignments with researchers to help better understand impact of LLMs with data
  - (detailed announcement later today)

# Today

- Discussion: Experiences from Lecture 7
- Discussion: Reading questions for Lecture 8
- Lecture
  - Spec-Driven Development
- In-Class Activity

# Discussion: Experiences from Lecture 6 Activity

- What types of questions was the LLM able to answer?
- What types of questions was the LLM not able to answer?
  - What did it do instead?
- In what situations do you feel LLM can be helpful in answering questions?

# Discussion: Reading questions for Lecture 7

- What questions did you have from readings for Lecture 7
  - Discuss questions & possible answers in group of 3 or 4
  - Come back with 1 question you want to discuss w/ whole class

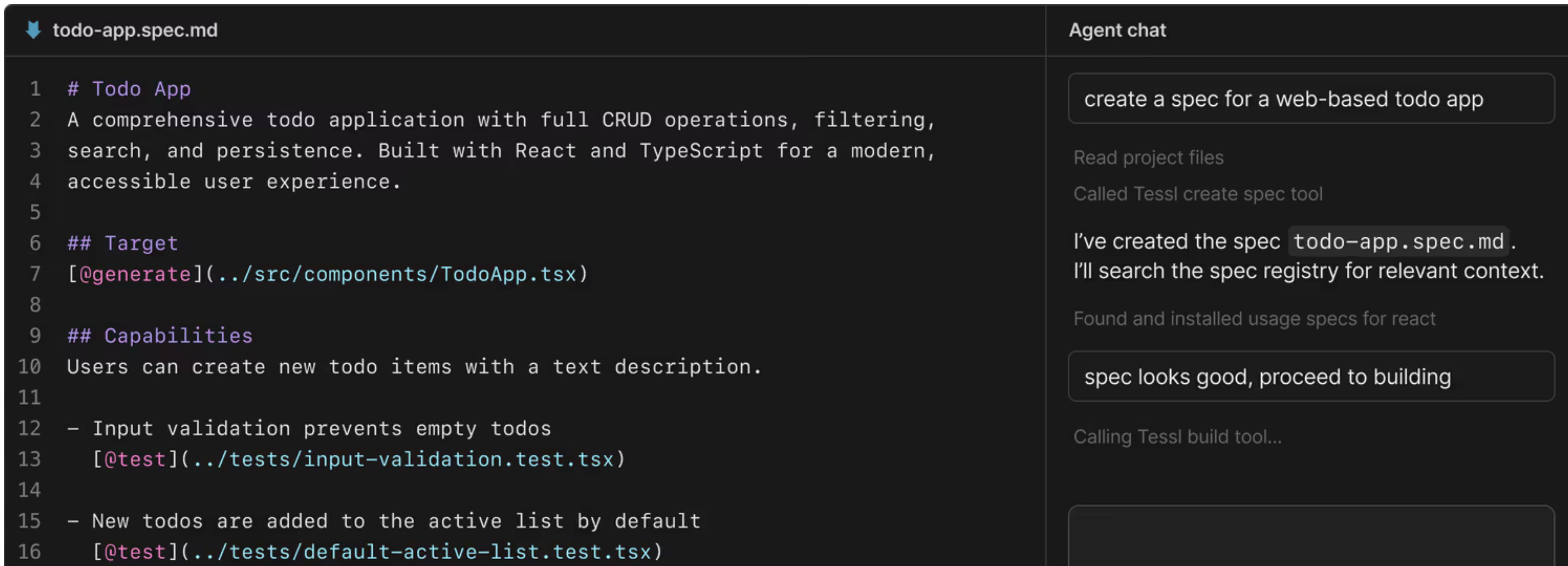
# Spec-Driven Development

# Motivation

- Agentic LLMs can generate lots of code
- But intent lost: instruct the LLM how to do something in chat, only see the code at the end
- Is there a better way?

# Spec Driven Development: Key Idea

- Capture what to do in a spec
- Rather than describe requirements through agents conversations, describe requirements in a doc
- Requirements doc becomes a "spec"
- Agent might even work with you to help write this spec from a high level plan



The image shows a terminal window on the left and an 'Agent chat' interface on the right. The terminal window displays the content of a file named 'todo-app.spec.md'. The 'Agent chat' interface shows a conversation with an agent named 'Tessl'.

**todo-app.spec.md**

```
1 # Todo App
2 A comprehensive todo application with full CRUD operations, filtering,
3 search, and persistence. Built with React and TypeScript for a modern,
4 accessible user experience.
5
6 ## Target
7 [@generate](..../src/components/TodoApp.tsx)
8
9 ## Capabilities
10 Users can create new todo items with a text description.
11
12 - Input validation prevents empty todos
13   [@test](..../tests/input-validation.test.tsx)
14
15 - New todos are added to the active list by default
16   [@test](..../tests/default-active-list.test.tsx)
```

**Agent chat**

create a spec for a web-based todo app

Read project files

Called Tessl create spec tool

I've created the spec `todo-app.spec.md`.

I'll search the spec registry for relevant context.

Found and installed usage specs for react

spec looks good, proceed to building

Calling Tessl build tool...

# Some Advantages

- Spec-centric understanding, moves focus from code ("how") to requirements ("what")
  - See exactly what is in the spec before it is built
  - Something missing or underspecified: change the spec
- Can easily have LLM generate tests simply by looking at requirements and trying to build corresponding tests for them
  - And can easily understand what is or is not currently tested by seeing the requirements
- Have confidence that spec really reflects the "what" of the code, as system generates the code from the spec & uses tests to check

# More ambitious (long term advantages)

- Portability: Can generate different versions of app for different platforms simply by recompiling spec for different platforms
- Compile time customizability: change preferences about what to optimize for (e.g., spend more \$\$\$ on cloud costs to make faster or fewer to spend less) w/ no need to change code
- End user programming: building requirements can be done w/ less CS knowledge (??)

# Some Challenges

- Asks a **lot** of LLM to successfully generate code from a spec in all cases
  - What happens when it fails or makes a poor choice? How does developer override the implementation and still trust the spec / design really reflects the code?
  - Or if overrides the implementation, and then changes the spec, how does the LLM know what to write, when everything is no longer captured in the spec?
  - If the developer still needs to debug something not working, how does that work?

# Current Status

- Big vision, not yet a reality
- Developer today already starting to do this by writing markdown files that capture the design, that are then fed to the LLM as part of the prompt
  - Achieves the goal of keeping specs on record for posterity
  - Can even use hooks to generate tests from these specs
  - But still have to (mostly) work with code
- Just starting to see tools that try to implement part of the vision
- Look at Amazon Kiro as an example (w/ a free tier!)
  - Differs in a number of important ways from longterm vision of spec-driven development to make more practical today

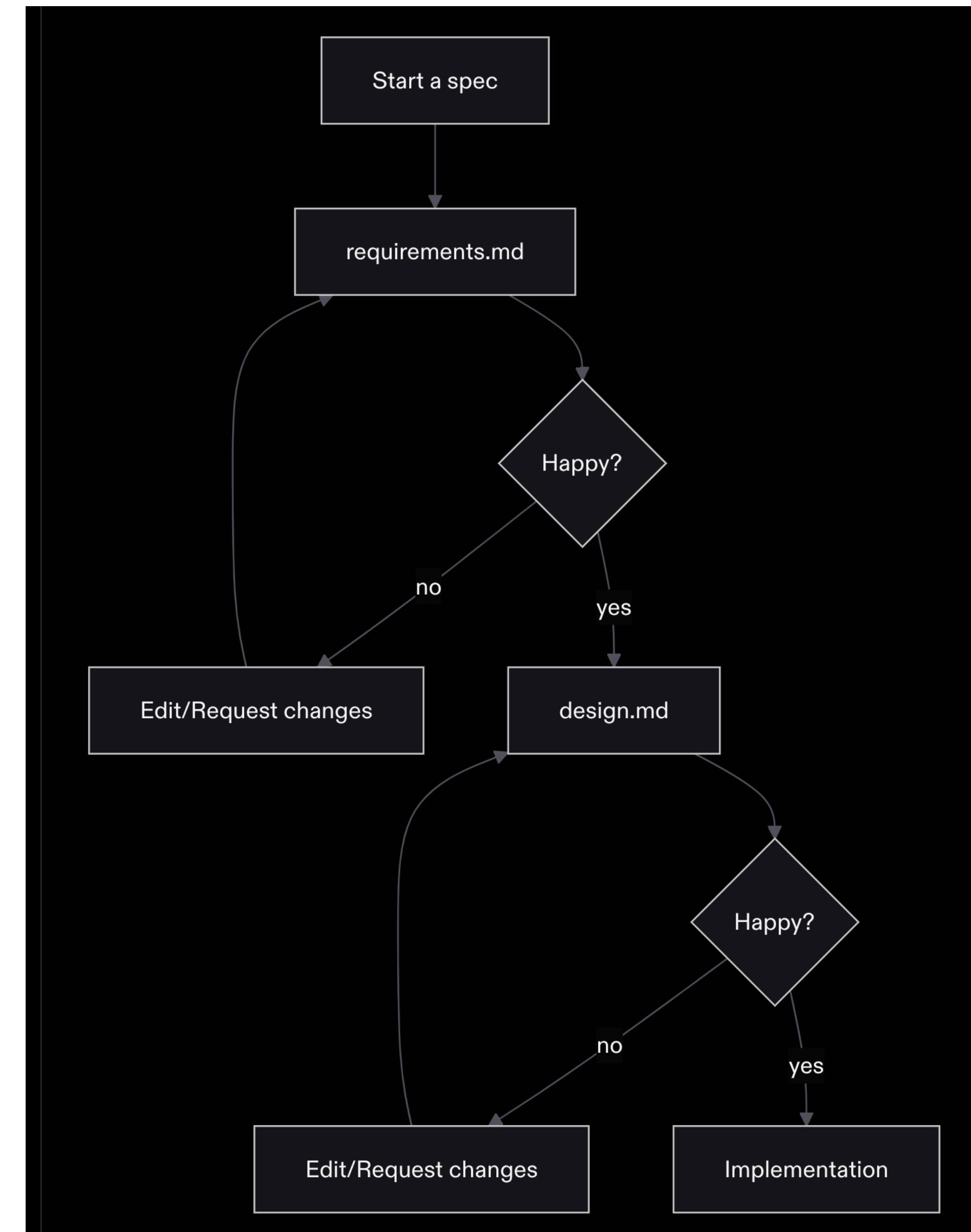
KIRO

# Specs Concepts

- `requirements.md` - Captures user stories and acceptance criteria in structured EARS notation
- `design.md` - Documents technical architecture, sequence diagrams, and implementation considerations
- `tasks.md` - Provides a detailed implementation plan with discrete, trackable tasks

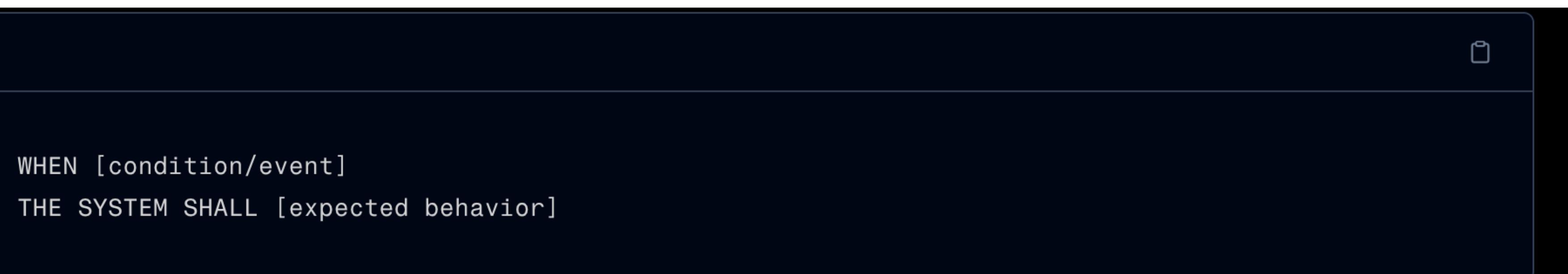
# Specs Workflow

- The workflow follows a logical progression with decision points between phases, ensuring each step is properly completed before moving to the next.
  - Requirements Phase (leftmost section): Define user stories and acceptance criteria in structured EARS notation
  - Design Phase (second section): Document the technical architecture, sequence diagrams, and implementation considerations
  - Implementation Planning (third section): Break down the work into discrete, trackable tasks with clear descriptions and outcomes
  - Execution Phase (rightmost section): Track progress as tasks are completed, with the ability to update and refine the spec as needed

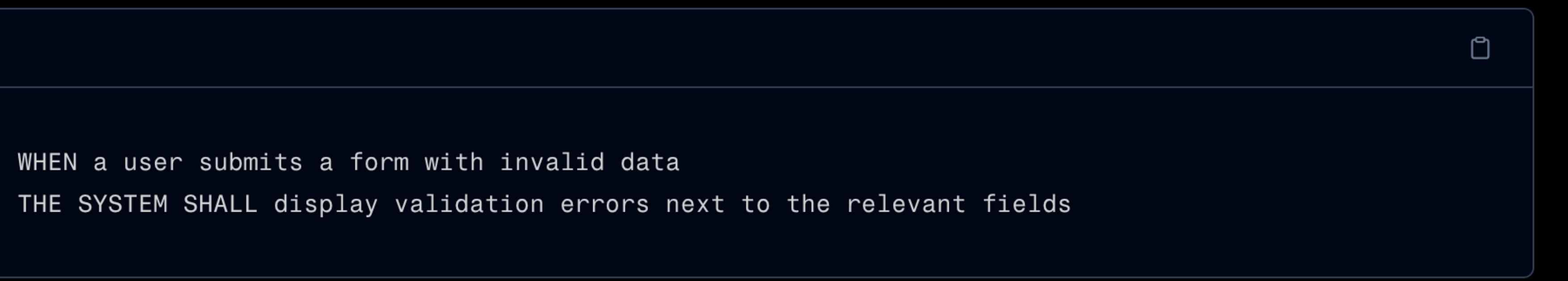


# Requirements docs

- Intended to use EARS (Easy Approach to Requirements Syntax) notation to provide a structured format
- Kiro helps format requirements into this structure



For example:

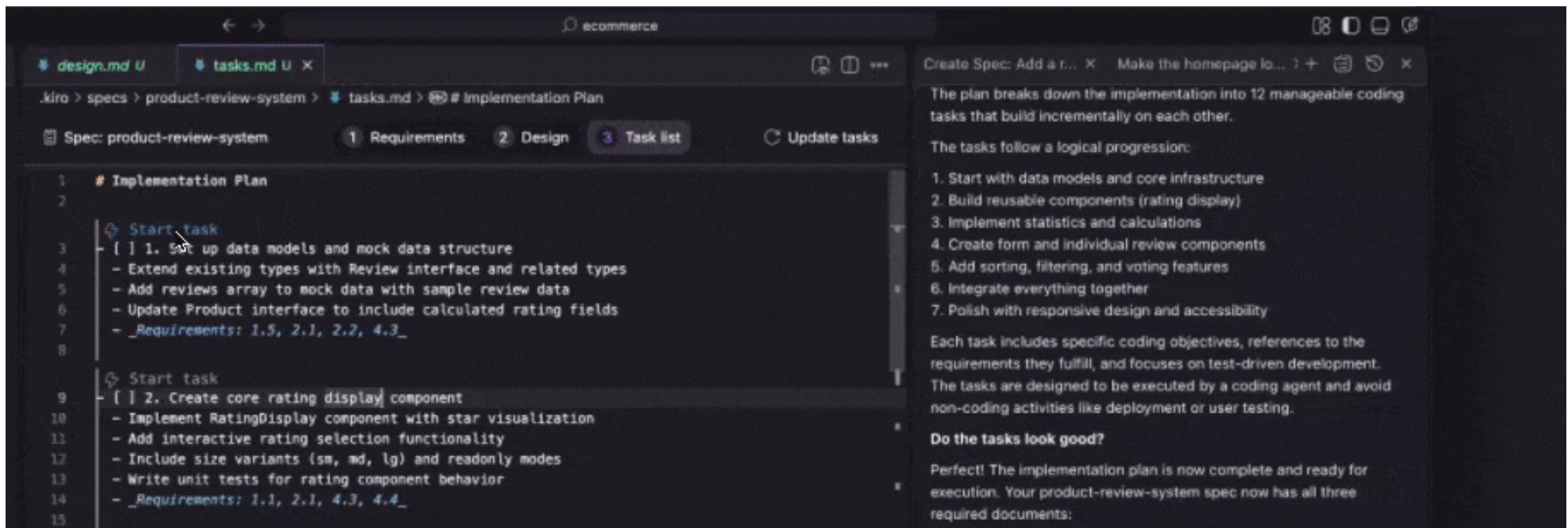


# Design Docs

- Document technical architecture, sequence diagrams, and implementation considerations
- Captures big picture of how the system will work, including the components and their interactions.

# Implementation plan

- Clearly defined tasks and subtasks with necessary resources and dependencies
- Shows real time status as tasks are completed



The screenshot shows a code editor with a dark theme. The title bar says "ecommerce". The left sidebar shows files: "design.md" (modified), "tasks.md" (modified), and "Create Spec: Add a r...". The main pane shows a file structure under ".kro > specs > product-review-system > tasks.md". The file content is an implementation plan:

```
1  # Implementation Plan
2
3  ↳ Start task
4  [ ] 1. Set up data models and mock data structure
5  - Extend existing types with Review interface and related types
6  - Add reviews array to mock data with sample review data
7  - Update Product interface to include calculated rating fields
8  - Requirements: 1.5, 2.1, 2.2, 4.3
9
10 ↳ Start task
11 [ ] 2. Create core rating display component
12 - Implement RatingDisplay component with star visualization
13 - Add interactive rating selection functionality
14 - Include size variants (sm, md, lg) and readonly modes
15 - Write unit tests for rating component behavior
16 - Requirements: 1.1, 2.1, 4.3, 4.4
```

The right side of the editor has a sidebar with the following text:

Create Spec: Add a r... × Make the homepage lo... : + ⌂ ⌂ ⌂ ⌂

The plan breaks down the implementation into 12 manageable coding tasks that build incrementally on each other.

The tasks follow a logical progression:

1. Start with data models and core infrastructure
2. Build reusable components (rating display)
3. Implement statistics and calculations
4. Create form and individual review components
5. Add sorting, filtering, and voting features
6. Integrate everything together
7. Polish with responsive design and accessibility

Each task includes specific coding objectives, references to the requirements they fulfill, and focuses on test-driven development.

The tasks are designed to be executed by a coding agent and avoid non-coding activities like deployment or user testing.

Do the tasks look good?

Perfect! The implementation plan is now complete and ready for execution. Your product-review-system spec now has all three required documents:

10 min break

# In-Class Activity

- In groups of 2 or 3, use Kiro requirements / design / implementation plan workflow to build a city simulator
  - Install Kiro (free plan w/ 30 days of free credits)
    - Mac ONLY -- need to find partner with a Mac
  - Build a city simulator
  - Goal: try to make it further into building a function city simulator than before
    - Pay attention to design to try to keep project manageable and head off poor choices
- Deliverables
  - Screen recording through Kaltura
    - Upload to OneDrive, turn on link sharing, share link in Lecture 8 activity submission on Canvas
    - Submit answers to questions on your experiences on Canvas (next slide)
- Aim to finish by 7:10pm today; Due tomorrow at 4:30pm

# Example City Simulator Requirements

(feel free to add additional requirements)

- Create transportation, power systems: roads, power lines, power plants; determine which zones have power or do not; calculate traffic along roads based on commuting & leisure activities; show traffic on roads
- Public transit system: trains, subways, with access through stations, that support commutes and leisure activities
- Build industrial shipping system, with support for connections to neighboring cities through roads & trains, supply chains connecting different industrial buildings; show supply chain industrial traffic on roads & trains
- Zone land for commercial, residential, industrial; Calculate value of zoned land every year, update building based on value changes
- Build crime system, with calculated crime scores based on building types & proximity to police, made visible on maps
- Build money system, with tax revenues & expenses based on buildings, roads, and other developments

# Questions to answer

- How many requirements were you able to use Kiro to complete?
- How did you make use of requirements, design, and implementation plan artifacts?
- In what ways were specs helpful as compared to vibe coding?
- In what ways did specs get in the way?
- Compared to working with a vibe coding environment, did you make more progress with specs? Why or why not?
- **Deliverable:** Submit through Canvas, at least a page