Spec-Driven Development

CS 691 / SWE 699
Fall 2025

GEORGE MASON UNIVERSITY

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

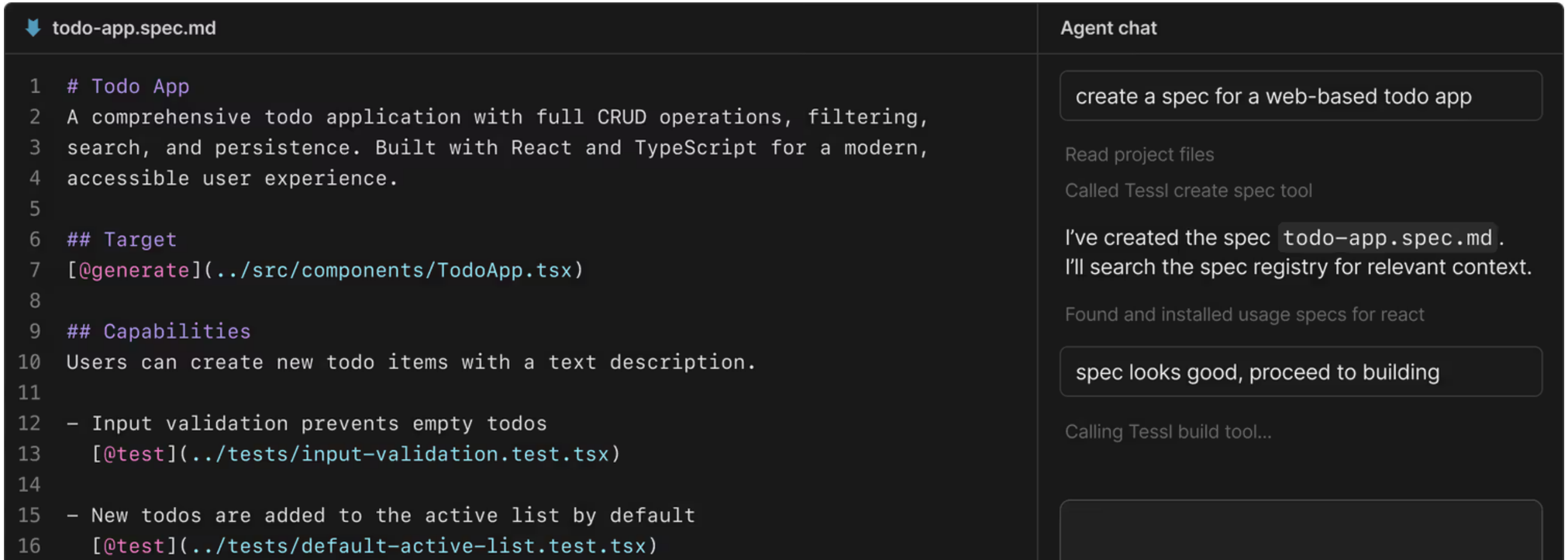
Spec-Driven Development

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



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 generates 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

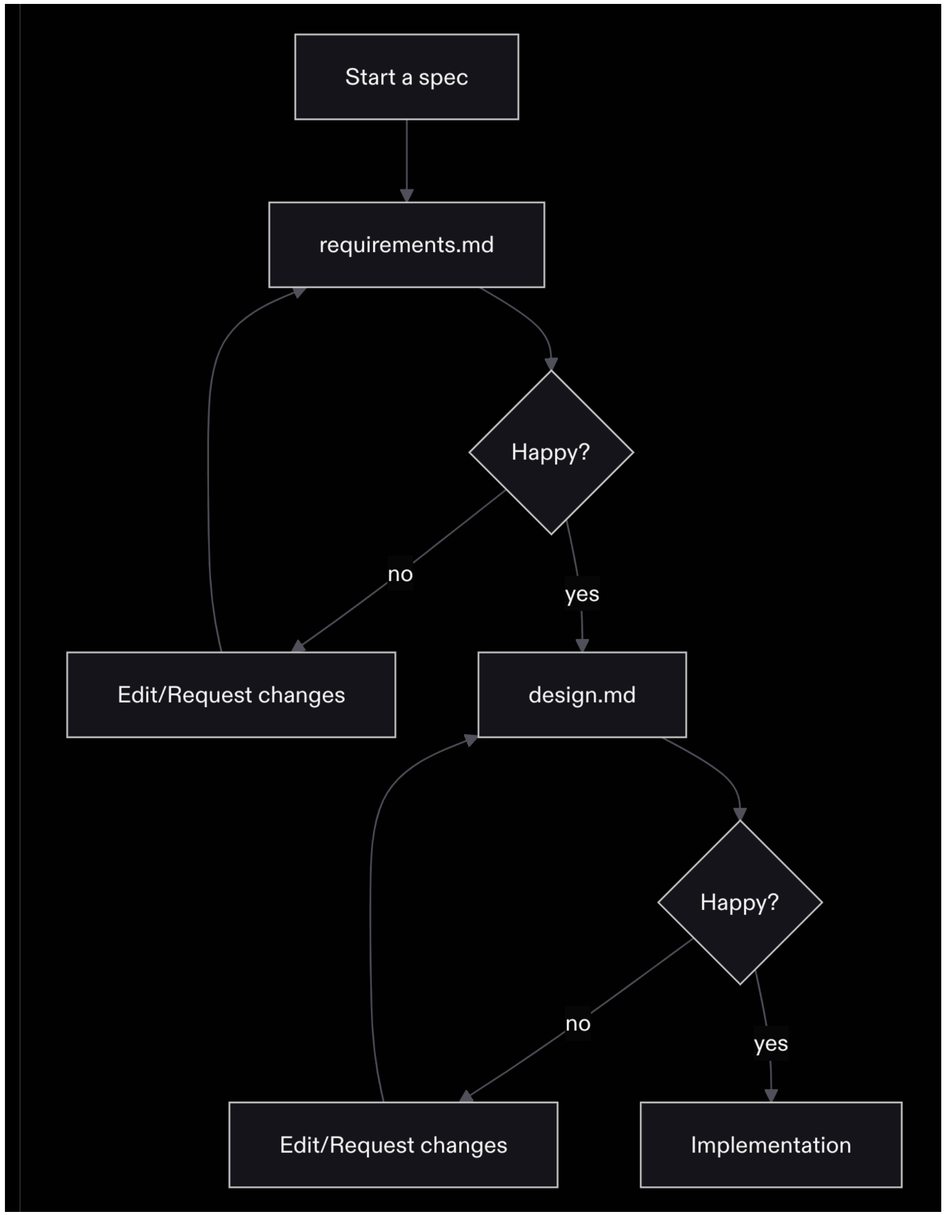
Specs Concepts

- 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

https://kiro.dev/docs/specs/concepts/

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



https://kiro.dev/docs/specs/concepts/

Requirements docs

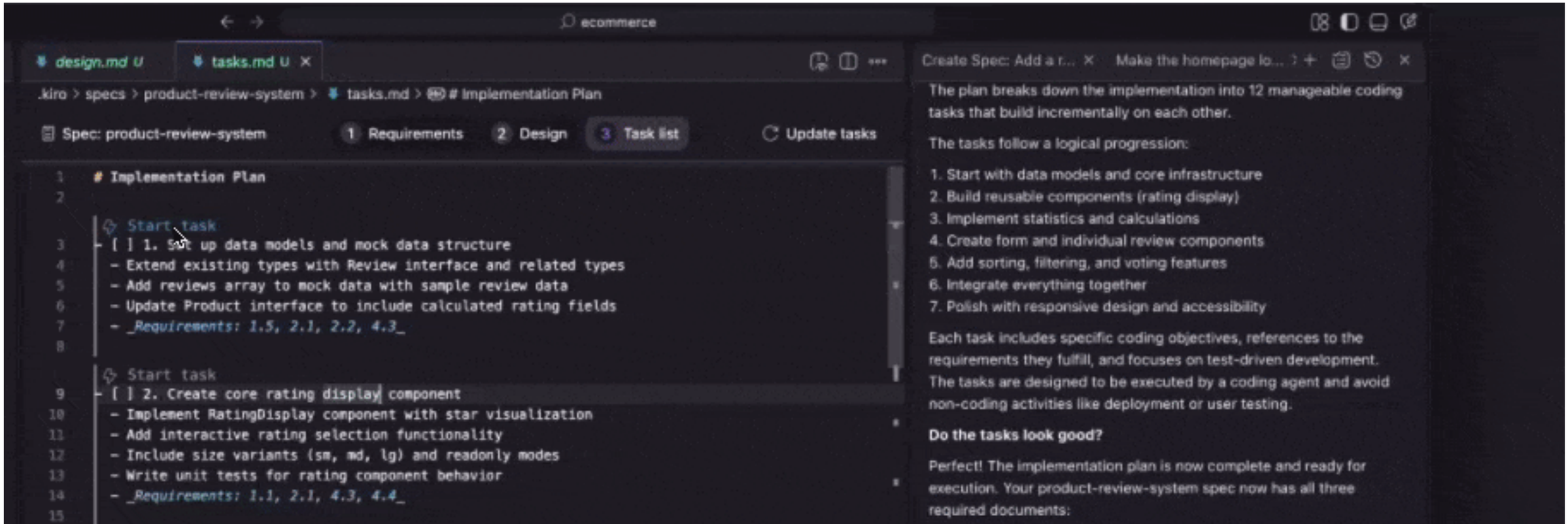
- Intended to use EARS (Easy Approach to Requirements Syntax) notation to provide a structured format
- Kiro helps format requirements into this structure

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



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