

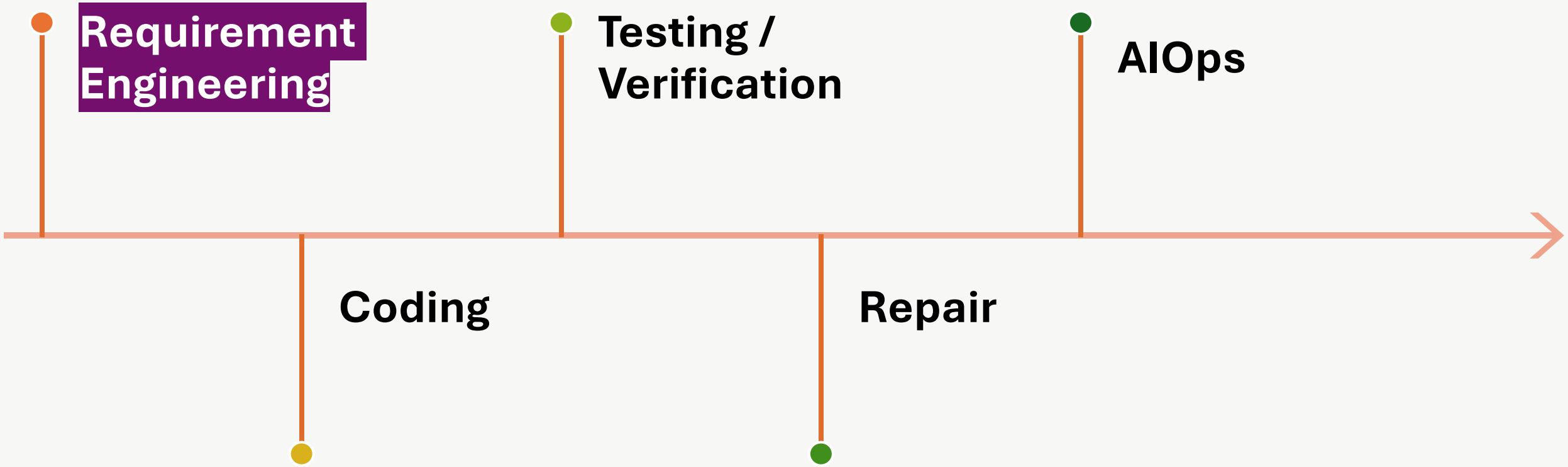
香港中文大學（深圳）
The Chinese University of Hong Kong, Shenzhen

CSC4801
AI-assisted Software Engineering

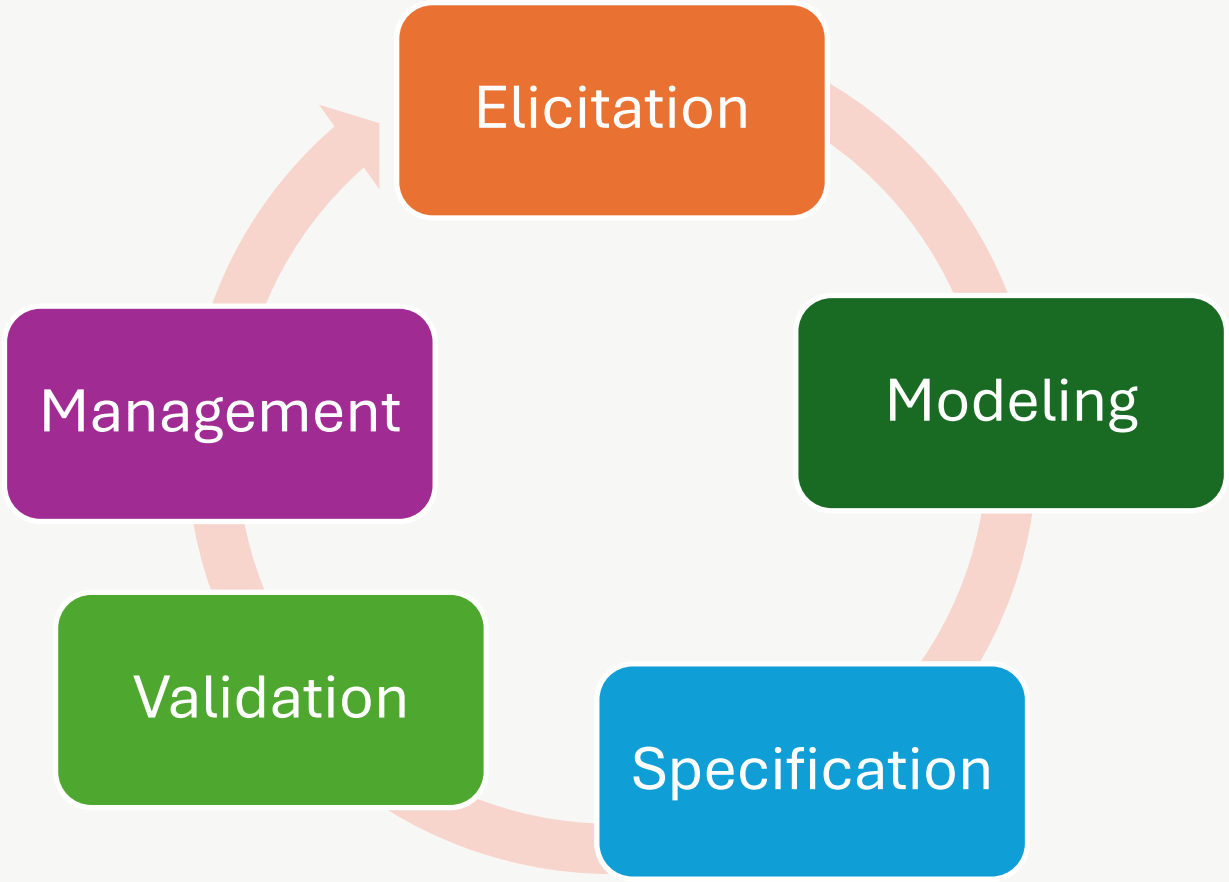
Lecture 07: Requirements Exercise

Instructor: Jinsheng Ba | 2026 Fall

AI-Assisted Software Engineering



Requirement Engineering

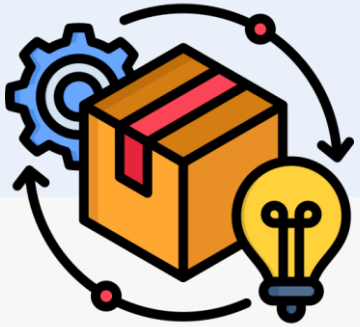


Strength

Thinking before action



Software engineering life cycle



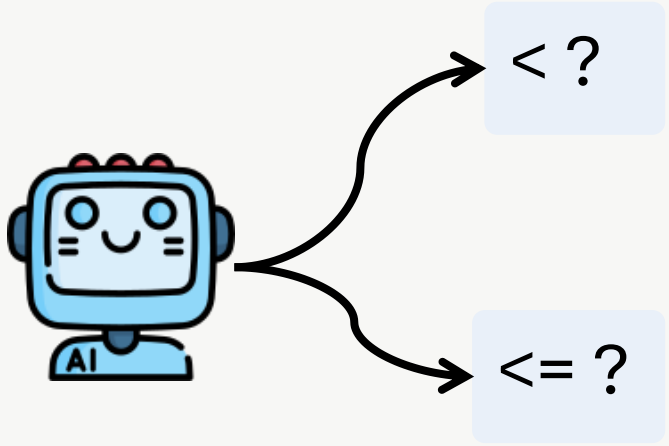
Navigating AI agents



Weakness

- Informal language incurs ambiguity

Insufficient user balance reject transactions.



But how to use them in coding agents?

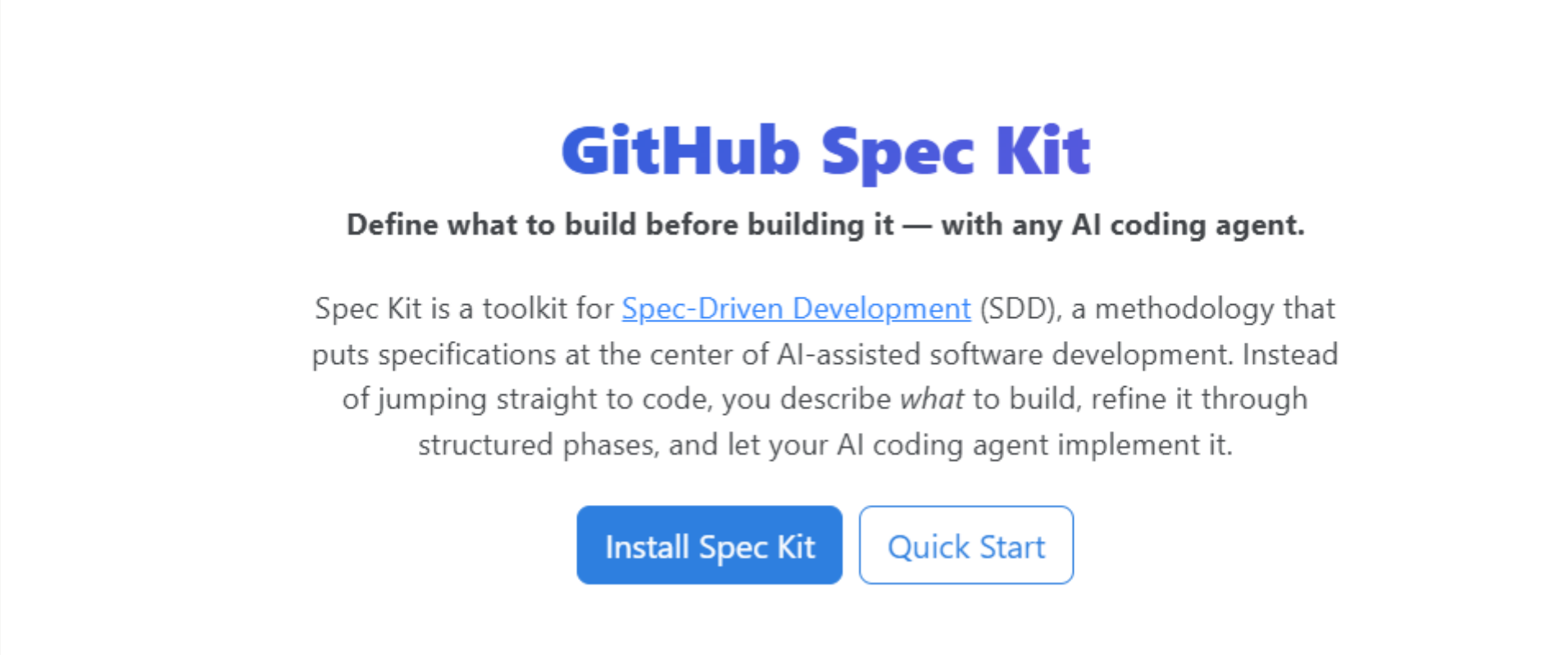
Spec-Driven Development

A methodology to write clear and detailed specification before implementation in LLM era.

Spec-Driven Development

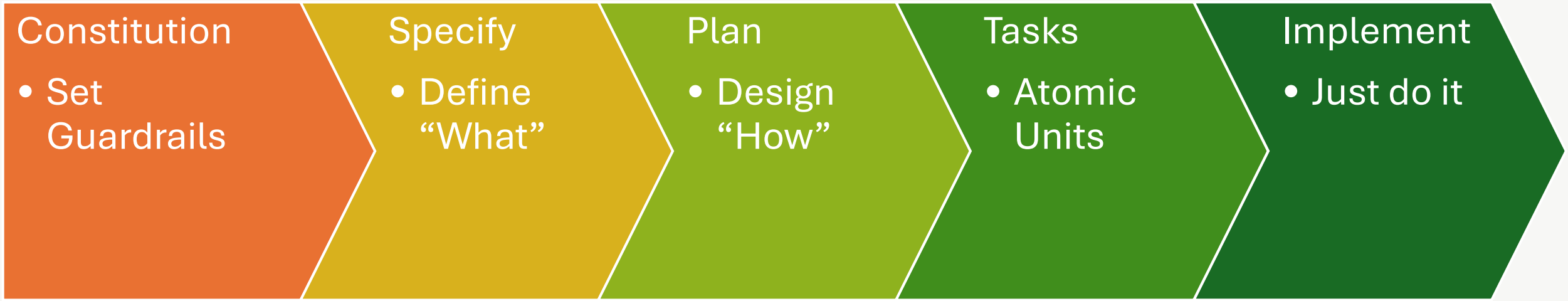
Argument: specifications are more important than the code.

GitHub Spec Kit



https://github.github.com/spec-kit/

The SDD Lifecycle



1. The Constitution

- Setting the principles for your project
  - Technical stack & architecture
  - Security & compliance standard
  - Performance benchmark
- Persistent in the project folder
  - “constitution.md”

Example

/speckit.constitution This project follows a "Library-First" approach. All features must be implemented as standalone libraries first. We use TDD strictly. We prefer functional programming patterns.

```
## Constitution Check

*GATE: Must pass before Phase 0 research. Re-check after Phase 1 design.*

- [ ] Is the feature implemented as a standalone library? (Library-First)
- [ ] Are we prepared to write tests first and ensure they fail? (TDD Strictly)
- [ ] Are we using functional programming patterns in the design?
```

.specify/templates/plan-template.md

2. Defining the Spec

Outcomes

What should the system achieve when the work is finished?



Boundaries

Define what is in-scope and explicitly out-of-scope to prevent agent scope-creep.



Constraints

List prior decisions and technical limits that the AI agent must respect.

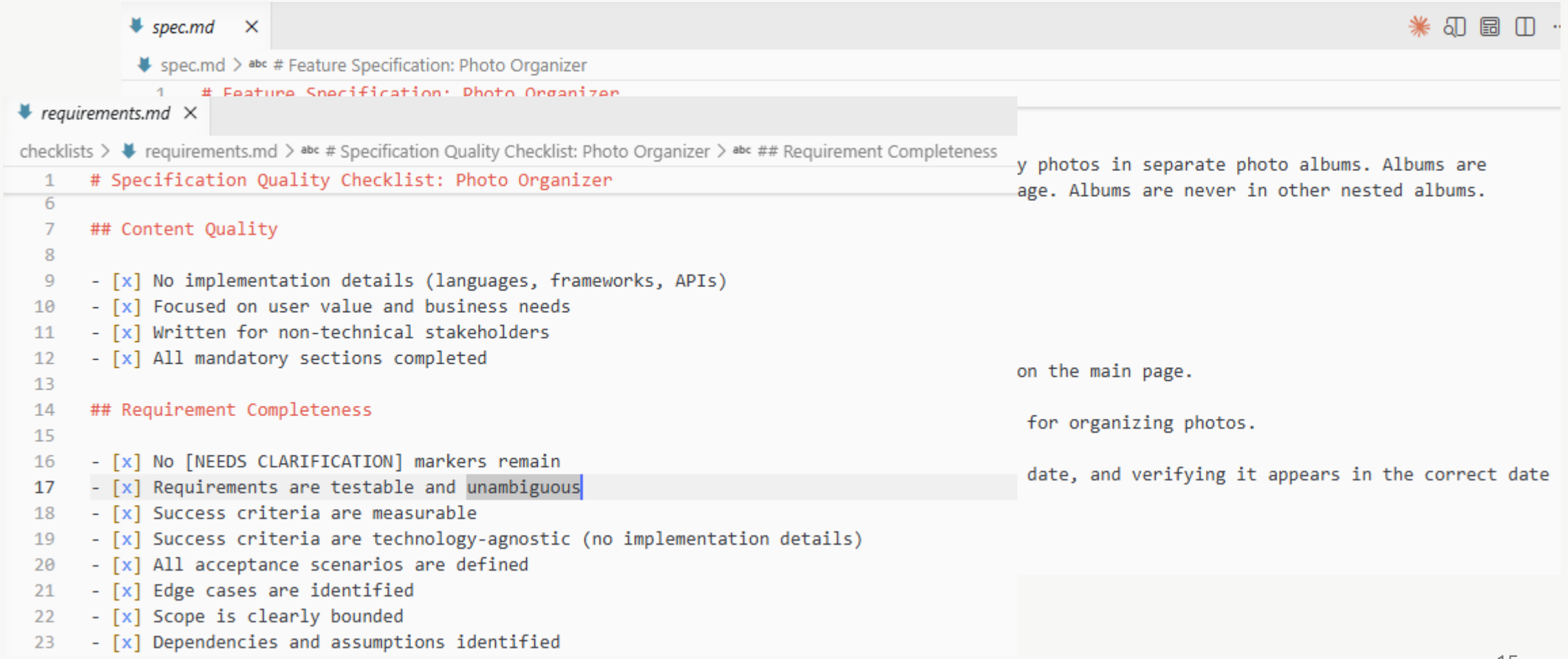


Example

/speckit.specify Build an application that can help me organize my photos in separate photo albums. Albums are grouped by date and can be re-organized by dragging and dropping on the main page. Albums are never in other nested albums. Within each album, photos are previewed in a tile-like interface.

.specify/templates/plan-template.md

Example


spec.md
spec.md > # Feature Specification: Photo Organizer
requirements.md
checklists > requirements.md > # Specification Quality Checklist: Photo Organizer > ## Requirement Completeness
Specification Quality Checklist: Photo Organizer
Content Quality
- [x] No implementation details (languages, frameworks, APIs)
- [x] Focused on user value and business needs
- [x] Written for non-technical stakeholders
- [x] All mandatory sections completed
Requirement Completeness
- [x] No [NEEDS CLARIFICATION] markers remain
- [x] Requirements are testable and unambiguous
- [x] Success criteria are measurable
- [x] Success criteria are technology-agnostic (no implementation details)
- [x] All acceptance scenarios are defined
- [x] Edge cases are identified
- [x] Scope is clearly bounded
- [x] Dependencies and assumptions identified
y photos in separate photo albums. Albums are
age. Albums are never in other nested albums.
on the main page.
for organizing photos.
date, and verifying it appears in the correct date

3. The Technical Plan

- The plan.md acts as the bridge between business intent and raw code. It defines:
  - Data models and schemas
  - API contracts and endpoints
  - Component architecture
  - Tech stack

Example

/speckit.plan The application uses Vite with minimal number of libraries. Use vanilla HTML, CSS, and JavaScript as much as possible. Images are not uploaded anywhere and metadata is stored in a local SQLite database.

.specify/templates/plan-template.md

Example

plan.md

plan.md

```
# Implementation Plan: Photo Organizer

**Branch**: `001-photo-organizer` | **Date**: 2026-06-25 | **Spec**: [specs/001-photo-organizer/spec.md](file:///zdata/jinsheng/ast_course/csc4801/test/specs/001-photo-organizer/spec.md)

**Input**: Feature specification from `/specs/001-photo-organizer/spec.md`

## Summary

Build a photo organization application allowing users to group photos into flat albums by date and reorganize them via drag-and-drop on the main page. The application will use Vite, Vanilla HTML/CSS/JS, and store metadata in a local SQLite database.

## Technical Context

**Language/Version**: JavaScript (ES6+), HTML5, CSS3, Node.js

**Primary Dependencies**: Vite, `better-sqlite3` (or equivalent Node SQLite library)

**Storage**: Local SQLite database for metadata. Local file system for images.

**Testing**: `vitest` (for TDD of library and UI)

**Target Platform**: Desktop web browser running against a local server

**Project Type**: Web Application with a local backend / library
```

4. Checklist

- Generates a custom quality checklist — "unit tests for your requirements" — to confirm the spec is complete, clear, and consistent before you break the work down.

Example

```
# Unit Tests for Requirements: English

**Purpose**: Verify requirements writing quality for the "Organize Photo Albums" feature.
**Created**: 2026-10-10
**Feature**: specs/001-organize-photo-albums/spec.md

**Review Ownership**: This checklist is a reviewer-owned artifact. Items are left unchecked for the reviewer to mark when satisfied.

## Requirement Completeness

- [ ] CHK001 - Are all primary user journeys (create/view albums, reorder albums, photo preview) documented? [Completeness, Spec §User Scenarios]
- [ ] CHK002 - Are edge cases (missing timestamps, large albums, offline reordering) explicitly defined? [Completeness, Spec §Edge Cases]
- [ ] CHK003 - Are non-functional requirements (performance targets for thumbnails) specified? [Completeness, Spec §Success Criteria]
- [ ] CHK004 - Is the fallback behavior for photos without timestamps specified and actionable? [Completeness, Spec §FR-006]

## Requirement Clarity

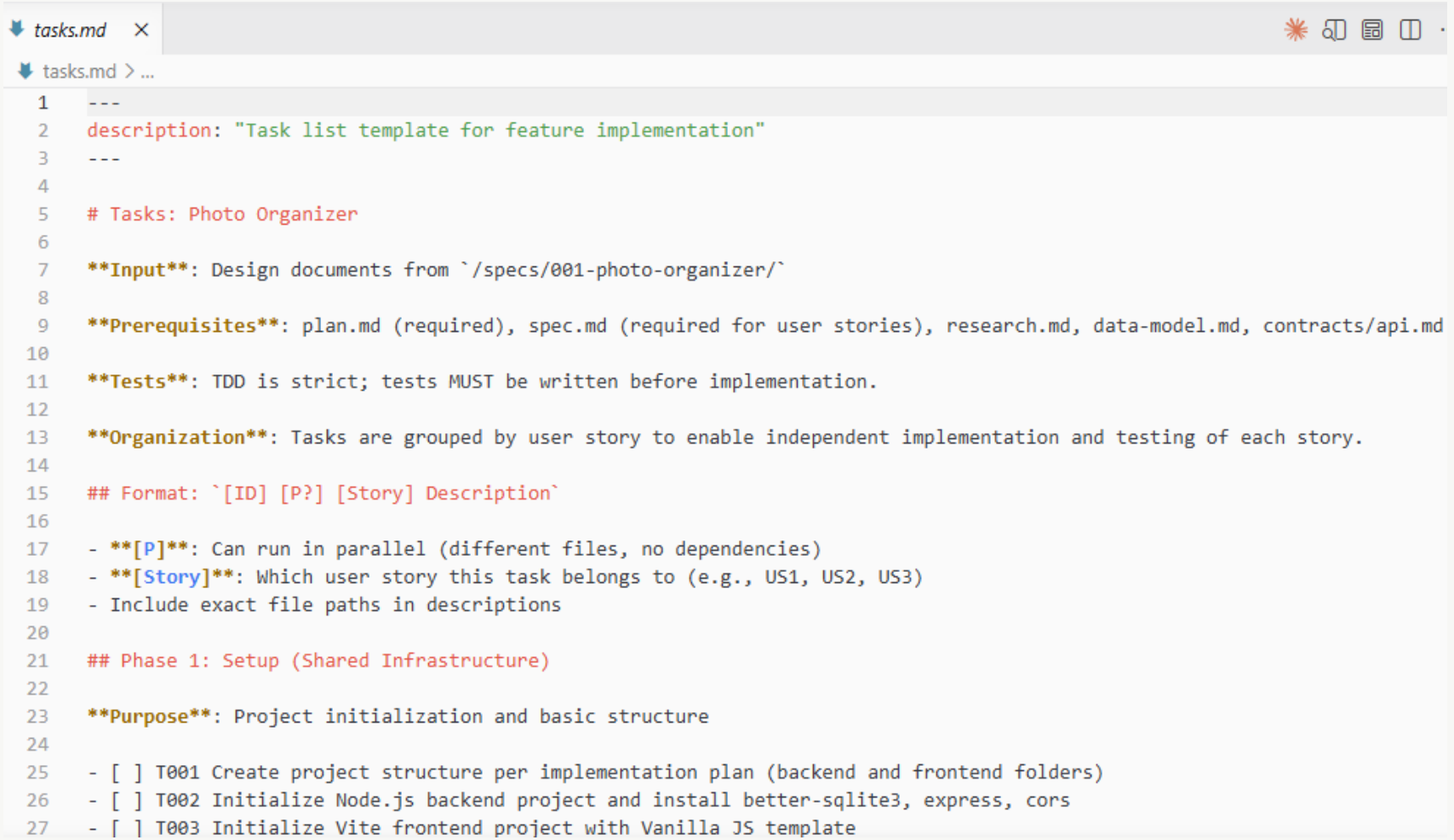
- [ ] CHK005 - Is the grouping granularity (by day) stated unambiguously and is alternative grouping noted as out-of-scope or a feature toggle? [Clarity, Spec
§Assumptions]
- [ ] CHK006 - Is "reorder persists" quantified with a persistence scope (per-user, per-device, server)? [Clarity, Spec §FR-007]
- [ ] CHK007 - Are tile preview behavior and lazy-loading expectations described clearly enough to drive acceptance tests? [Clarity, Spec §User Story 4]
```

5. Tasks

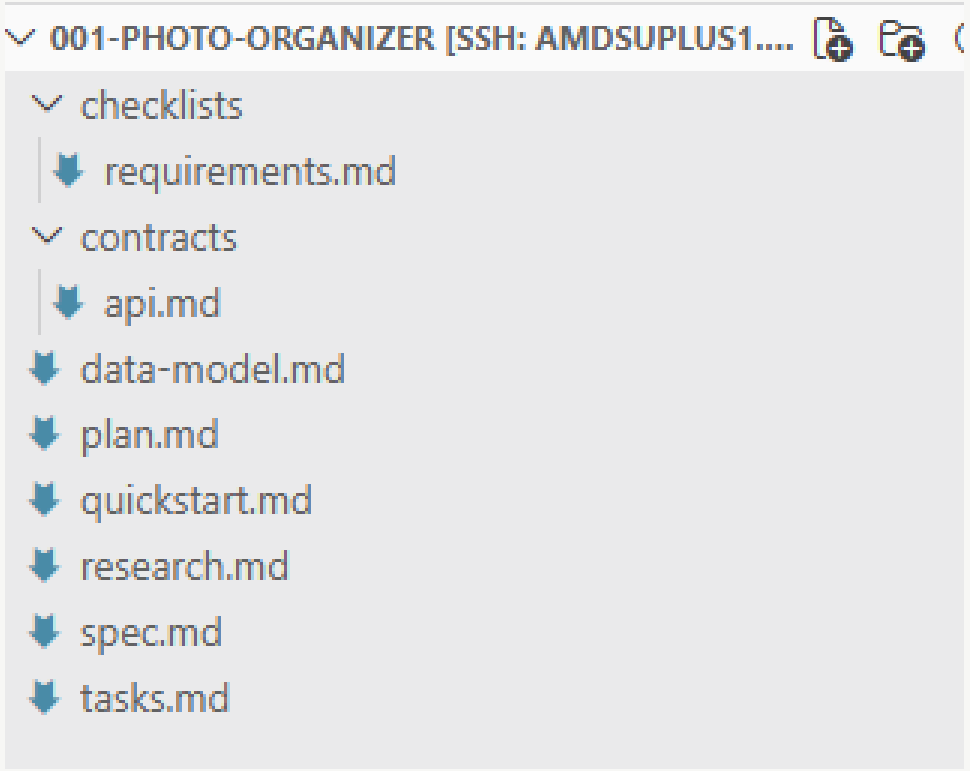
- Breaking the plan into atomic units.
- This ensures the AI agent focuses on one testable change at a time, verified against the original spec at every step.

Example

/speckit.tasks



Example: All Specifications



6. Analyze

- Reports conflicts, gaps, and ambiguities across spec.md, plan.md, and tasks.md.

Example

- **Specification Analysis Report — Organize Photo Albums**

  Feature: specs/001-organize-photo-albums
  Artifacts analyzed: spec.md, plan.md, tasks.md
  Constitution: .specify/memory/constitution.md (authoritative)

  Summary metrics
  - Total FRs found: 9 (FR-001 … FR-009)
  - Total SCs found: 4 (SC-001 … SC-004)
  - Total tasks: 24 (T001–T015, T020–T023, T030–T032, T044, T900–T903)
  - Requirements with ≥1 mapped task: 9 / 9 (100%) — see notes about partial coverage below
  - Duplication findings: 0
  - Ambiguity findings: 3
  - Critical issues (constitution violations / blocking gaps): 2
  - High-severity issues: 1
  - Medium/Low: several (see table)

  Notes on approach
  - I mapped each FR / SC to tasks by keyword and by user-story labels in tasks.md.
  - I treated constitution rules (MUST/SHOULD) as non-negotiable constraints; any deviation is flagged CRITICAL.
  - This is read-only analysis; I did not modify any files.

7. Implementation

tasks.md

```
---
description: "Task list template for feature implementation"
---

# Tasks: Photo Organizer

**Input**: Design documents from `/specs/001-photo-organizer/`

**Prerequisites**: plan.md (required), spec.md (required for user stories), research.md, data-model.md, contracts/api.md

**Tests**: TDD is strict; tests MUST be written before implementation.

**Organization**: Tasks are grouped by user story to enable independent implementation and testing of each story.

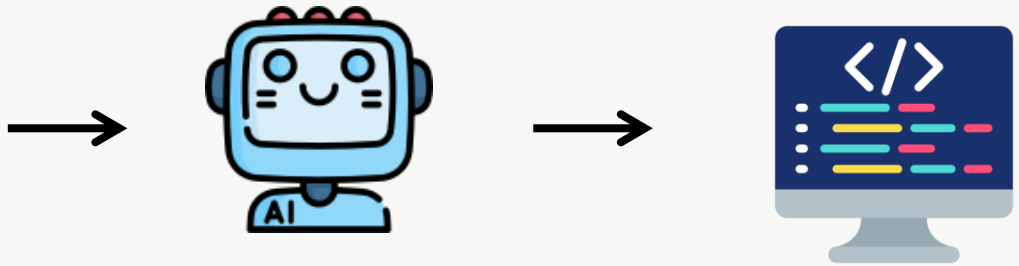
## Format: `[ID] [P?] [Story] Description`

- **[P]**: Can run in parallel (different files, no dependencies)
- **[Story]**: Which user story this task belongs to (e.g., US1, US2, US3)
- Include exact file paths in descriptions

## Phase 1: Setup (Shared Infrastructure)

**Purpose**: Project initialization and basic structure

- [ ] T001 Create project structure per implementation plan (backend and frontend folders)
- [ ] T002 Initialize Node.js backend project and install better-sqlite3, express, cors
- [ ] T003 Initialize Vite frontend project with Vanilla JS template
```



How is SDD Implemented?

```
$ tree .agents/skills
.agents/skills
├── speckit-agent-context-update
│   └── SKILL.md
├── speckit-analyze
│   └── SKILL.md
├── speckit-checklist
│   └── SKILL.md
├── speckit-clarify
│   └── SKILL.md
├── speckit-constitution
│   └── SKILL.md
├── speckit-converge
│   └── SKILL.md
├── speckit-implement
│   └── SKILL.md
├── speckit-plan
│   └── SKILL.md
├── speckit-specify
│   └── SKILL.md
├── speckit-tasks
│   └── SKILL.md
└── speckit-taskstoissues
    └── SKILL.md

11 directories, 11 files
```

Skills

Demo

Exercise

- You can implement SDD by yourself.

Goal

- Use Spec Kit to develop to analyze requirements
- Find inconsistencies in a set of requirements

Write the Constitution

- Create a constitution.md for your project.
  Include:
  Technical stack & architecture principles
  Security & compliance standards
  Performance benchmarks
  Coding conventions

```
# constitution.md

## Tech Stack
- Frontend: React + TypeScript
- Backend: Node.js + Express
- Database: PostgreSQL

## Principles
- Library-First: all features as standalone modules
- Test-Driven: every feature has tests before implementation
- API-First: define OpenAPI spec before coding

## Security
- No hardcoded credentials
- Input validation on all endpoints
- HTTPS only
```

Write the Specification

- Using your extracted requirements, write spec.md.
  Define:
  Outcomes: What should the system achieve?
  Boundaries: In-scope vs. out-of-scope
  Constraints: Technical limits, deadlines

```
# spec.md

## Outcomes
- Students can search, filter, and register for courses
- Professors can manage course rosters
- Admins can configure registration periods

## Boundaries
In scope: Course search, registration, roster management
Out of scope: Payment processing, grades, attendance

## Constraints
- Must support 1000 concurrent users
- Registration period is configurable per semester
- No third-party auth (use university SSO)
```

Create the Technical Plan

- Bridge business intent to code.
  Define:
  Data models / schemas
  API contracts / endpoints
  Component architecture
  Task breakdown

```
# plan.md

## Data Models
Course: { id, name, code, capacity,
professor_id }
Student: { id, name, email, major }
Enrollment: { student_id, course_id, status }

## API Endpoints
GET  /api/courses?search=&major=
POST /api/courses/:id/register
GET  /api/professors/:id/roster

## Components
CourseSearch → CourseList → RegistrationForm
ProfessorDashboard → RosterView
```

Requirement Verification

- Read the following requirements for a "Smart Home Controller":

  REQ-1: The system shall control all lights in the house.
  REQ-2: The system shall respond to voice commands.
  REQ-3: The system shall learn user preferences over time.
  REQ-4: The system shall work without internet connection.
  REQ-5: The system shall integrate with Alexa and Google Home.
  REQ-6: The system shall be affordable (under $100).

  Find as many issues as you can.

Categorize the Issues

- Group the issues you found into categories:

  Ambiguity: "affordable" — affordable to whom?
  Conflict: REQ-4 (no internet) vs REQ-5 (Alexa needs internet)
  Missing: What happens during power outage?
  Unmeasurable: "learn user preferences" — how much? how fast?
  Incomplete: No security requirement (what if someone hacks it?)

Rewrite the Requirements

- Fix each issue by making requirements:
  Specific: replace vague terms with measurable criteria
  Consistent: resolve conflicts explicitly
  Complete: add missing requirements
  Testable: every requirement must have a clear test

```
# Before (ambiguous)
REQ-6: The system shall be affordable (under $100).

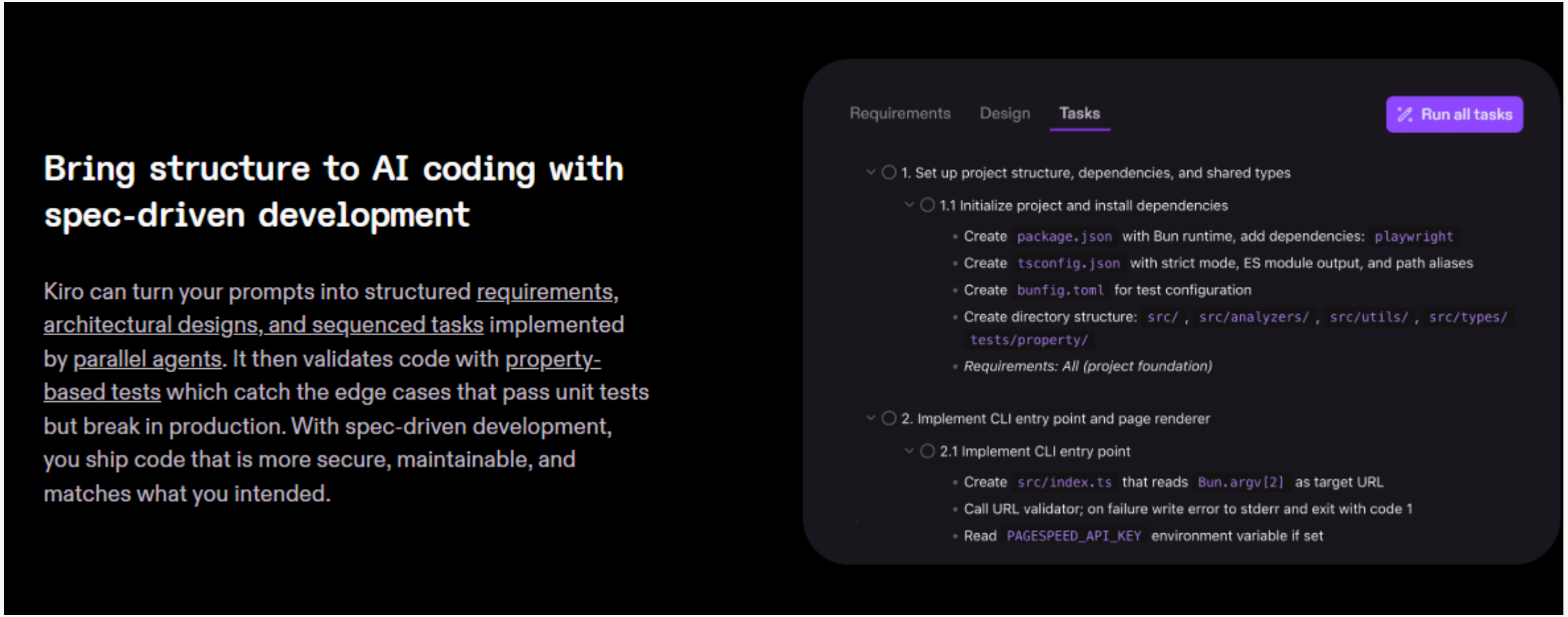
# After (specific + testable)
REQ-6: The system hardware + software total cost
shall not exceed $100 USD at retail pricing
as of the product launch date.
Test: Bill of materials review, price verification.
```

Verify Your Rewritten Requirements

- Run the LLM verification again on your rewritten set.
  Did you introduce new issues?
  Are all original issues resolved?
  This is the verification loop: write → check → fix → re-check.

Kiro

Kiro, a coding agent that deeply integrate SDD workflow.



https://kiro.dev/



香港中文大學（深圳）
The Chinese University of Hong Kong, Shenzhen

Questions

Next class: Coding I