Cloud Contraptions
TRAINING & CONSULTING

To discuss this course and customizations:
Call: +1 434-509-6890 or Email: sales@cloudcontraptions.com

Spec-Driven Development with AI Coding Agents: Spec Kit, Kiro, and OpenSpec

Class Duration

14 hours of live training delivered over 2-3 days to accommodate your scheduling needs.

Student Prerequisites

- Professional software development experience
- Working use of at least one AI coding agent (GitHub Copilot, Claude Code, Cursor, Codex, Kiro, or Google Antigravity)
- Comfort with the command line, Git, and pull requests
- Experience writing or reviewing requirements is helpful

Target Audience

Software engineers, tech leads, architects, and the product owners and analysts who write requirements. Platform teams that want a standard way to plan and review AI-generated code will also benefit. Teams adopting the full AWS method can continue with AI-Driven Development Life Cycle (AI-DLC), and Kiro teams can go deeper in Spec-Driven Agentic Coding with Kiro.

Description

Spec-driven development makes a reviewed specification the source of intent for an AI coding agent. The agent’s plan, tasks, and code are then checked against that spec. This course teaches the practice with GitHub Spec Kit, which runs inside dozens of coding agents. Students write specs and EARS acceptance criteria, run the Spec Kit workflow, and use its checks to find gaps. The course then compares Kiro specs, OpenSpec, BMAD, and Tessl, and covers review gates in pull requests and CI.

Learning Outcomes

- Distinguish spec-first, spec-anchored, and spec-as-source approaches.
- Write acceptance criteria precise enough for an agent to implement.
- Write requirements in EARS notation.

Copyright 2026 Cloud Contraptions LLC – All Rights Reserved

To discuss this course and customizations:
Call: +1 434-509-6890 or Email: sales@cloudcontraptions.com

- Author and amend a project constitution.
- Run the GitHub Spec Kit workflow from constitution to convergence.
- Use Spec Kit's clarify, checklist, and analyze steps to catch gaps.
- Propose changes to existing code with OpenSpec delta specs.
- Choose between Spec Kit, Kiro specs, OpenSpec, and BMAD for a project.
- Use spec-driven workflows with common coding agents.
- Detect spec drift and keep specs current.
- Add spec checks to pull requests and CI.

Training Materials

Comprehensive courseware is distributed online at the start of class. All students receive a downloadable MP4 recording of the training.

Software Requirements

Git, Python 3.11 or later with `uv` and the Spec Kit `specify` CLI, Node.js 22.12 or later with the OpenSpec CLI, a GitHub account with a sample repository, and an active subscription or API key for at least one supported coding agent (GitHub Copilot, Claude Code, Cursor, Codex CLI, Kiro, or Antigravity).

Training Topics

Spec-Driven Development Foundations

- Where prompt-first workflows break down
- Requirements lost in chat history
- Specs as versioned contracts
- Spec-first, spec-anchored, and spec-as-source
- Relation to TDD, BDD, and ADRs
- Matching ceremony to the change

Writing Specs and Acceptance Criteria

- Separating what from how
- User stories and functional requirements
- Measurable success criteria
- Constraints and out-of-scope boundaries
- Explicit open questions
- EARS requirement patterns

Copyright 2026 Cloud Contraptions LLC – All Rights Reserved

- Event-driven and state-driven requirements
- Unwanted behavior and optional features
- WHEN/THEN scenarios
- Mapping criteria to tests
- Vague and oversized specs

The Project Constitution

- Non-negotiable project principles
- Quality, testing, and security rules
- Creating a constitution with `/speckit.constitution`
- Versioning and amendments
- The Sync Impact Report
- Constitution Check gates
- Grounding a constitution in an existing repository
- Constitutions vs. `AGENTS.md` and steering

GitHub Spec Kit Setup and Project Structure

- Installing the `specify` CLI
- `specify init --integration`
- Adding Spec Kit with `--here`
- The `.specify/` directory
- Feature directories under `specs/`
- Supported agent integrations
- Agent-specific command spellings
- Switching agents with `specify integration switch`
- Extensions, presets, and template overrides
- The opt-in `git` extension

The Spec Kit Workflow: Specify Through Analyze

- Writing the spec with `/speckit.specify`
- `[NEEDS CLARIFICATION]` markers
- The built-in requirements checklist
- Resolving gaps with `/speckit.clarify`
- Planning with `/speckit.plan`
- Research, data model, and contracts
- Custom checklists with `/speckit.checklist`
- Phased task lists with `/speckit.tasks`

Copyright 2026 Cloud Contraptions LLC – All Rights Reserved

- Parallel task markers
- Consistency checks with `/speckit.analyze`

Implementation and Convergence

- Running tasks with `/speckit.implement`
- Correcting an agent that drifts
- Gap analysis with `/speckit.converge`
- The implement-converge loop
- Splitting large features into specs
- Workflows with `specify workflow run`
- The `bug` and `github` extensions

Keeping Specs Current

- Changing an existing feature
- New or revised specs
- Living specs and other persistence models
- How specs and code drift
- Catching drift in review
- Preserving decision records
- What stays in version control

Kiro Specs Compared

- Feature specs in `.kiro/specs/`
- Requirements in EARS notation
- Requirements-first vs. design-first
- Quick Spec and Plan sessions
- Bugfix specs
- Property-based correctness checks
- Running tasks in waves
- Steering vs. a Spec Kit constitution
- Spec Kit in Kiro CLI

OpenSpec for Changing Existing Code

- A brownfield-first model
- Current specs in `openspec/specs/`
- Change folders in `openspec/changes/`
- Proposals, designs, and tasks
- Delta specs

Copyright 2026 Cloud Contraptions LLC – All Rights Reserved

- Added, modified, removed, and renamed requirements
- SHALL requirements and WHEN/THEN scenarios
- Propose, apply, and archive
- The `/openspec-propose` skill
- Merging deltas with `openspec archive`
- `openspec validate --strict`

BMAD Planning and Spec-as-Source with Tessl

- BMAD Method agent personas
- Analyst, PM, and architect agents
- From PRD to architecture
- Epics and stories
- Sizing the planning track
- Feeding BMAD output to implementation
- Tessl and spec-as-source
- Tessl's skills registry
- Limits of spec-as-source

Choosing an Approach and Integrating Agents

- Comparing the four approaches
- Greenfield vs. brownfield fit
- Tool lock-in
- Choosing per type of change
- Spec Kit in Copilot and Claude Code
- Other agent integrations
- Markdown specs as the portable layer
- Specs within AWS AI-DLC

Review Discipline and CI Integration

- Review gates at every phase
- Who reviews specs, plans, and code
- Tracing changes to requirement IDs
- Spec diffs in pull requests
- Branch conventions for spec work
- Non-interactive runs in CI
- Structural spec validation
- Flagging behavior changes without spec updates

Copyright 2026 Cloud Contraptions LLC – All Rights Reserved

To discuss this course and customizations:
Call: +1 434-509-6890 or Email: sales@cloudcontraptions.com

- Rework and convergence metrics

Copyright 2026 Cloud Contraptions LLC – All Rights Reserved