

COURSE OUTLINE

Specs-Driven Development — From Vibe Coding to Reliable AI-Assisted Delivery

A practical program for turning business intent into context-rich specs, buildable tasks, executable tests, and reviewable code

DURATION	FORMAT		TRAINER
2 days	Instructor-led, hands-on, lab-driven, project-based		
DELIVERY			
Online Instructor Led Workshop · Onsite Program for your Teams			Muhammed Shakir

Why this course exists

This program turns AI-assisted software delivery from vague prompting and optimistic code generation into a disciplined, specification-driven engineering workflow.

AI coding tools can generate code quickly, but speed without context, constraints, decisions, and tests creates a new class of delivery risk: plausible code that solves the wrong problem, violates the existing architecture, misses edge cases, or cannot be reviewed confidently.

Participants learn how to prepare repository context, understand an unfamiliar codebase, generate and challenge functional and technical specs, express contracts in machine-readable formats, configure coding agents, generate tests, and audit implementation against the source of truth.

The course culminates in an end-to-end tool-led SDD lab: a team moves from a business brief through repository context, codebase understanding, functional and technical specifications, API and data contracts, tasks, tests, implementation, and review.

What makes this different

Most AI coding courses focus on writing better prompts or demonstrating one coding agent. **This is a delivery-system course: specifications become the control layer that connects business intent, repository context, architecture, implementation, testing, and review.** You will:

- **Diagnose vibe coding failure modes** — see how ambiguity, missing context, hallucination, and untested assumptions become wrong code.
-

Prepare context before asking for code — package the right repository files, map system relationships, and manage the context budget.

- **Write specs that agents can execute** — make requirements, rules, constraints, assumptions, edge cases, and acceptance criteria explicit.
- **Use clarification and critique loops** — make ambiguity visible before it becomes implementation rework.
- **Compare the modern SDD tool landscape** — understand how Spec Kit, Kiro, Claude Code, Codex, Cursor, Copilot, Aider, and OpenCode fit into delivery.
- **Make contracts machine-readable** — use OpenAPI, JSON Schema, DBML, Mermaid, state diagrams, and workflow contracts to reduce interpretation gaps.
- **Test and review against the spec** — turn acceptance criteria into validation assets and audit AI-generated code for correctness, security, architecture, and regression risk.

The through-line is a complete delivery pack that a human team and an AI coding agent can both understand, challenge, implement, and verify.

Skills You'll Walk Away With

- Explain why AI-assisted development needs specifications as a control layer.
- Distinguish a prompt, specification, plan, task, and test.
- Compare spec-first, chat-first, and IDE-agent development models.
- Select SDD tools that fit your team's software development lifecycle.
- Prepare AI-ready repository context with deliberate file selection and token-budget awareness.
- Extract architecture, modules, dependencies, call flows, and spec boundaries from an unfamiliar codebase.
- Generate functional specs from business notes and harden them manually.
- Detect ambiguity, conflicting rules, missing decisions, and hidden assumptions.
- Use clarification loops, decision tables, examples, counterexamples, and Given-When-Then scenarios.
- Apply a GitHub Spec Kit-style specification-to-plan-to-task workflow.
- Create persistent steering documents, hooks, project memory, and agent instruction files.
- Generate technical specs grounded in existing repository conventions.
- Express API, data, workflow, and state contracts in machine-readable formats.
- Specify AI features, model behavior, grounding, fallback, refusal, confidence, and human escalation.
- Generate acceptance, unit, API, UI, and model-behavior tests from specs.
- Review AI-generated code for traceability, architecture violations, security issues, and regression risk.
- Produce a reusable, AI-agent-ready delivery pack from a business brief.

Who should attend

This program is for teams adopting AI coding tools who want to improve delivery quality, reviewability, and consistency without slowing down useful experimentation.

- **Business analysts and product managers** — translating business intent into clear, testable requirements.
- **Developers and technical leads** — creating implementation-ready technical context and constraints.
- **QA engineers and testers** — turning specifications into executable validation assets.
- **Architects and engineering managers** — establishing conventions, guardrails, and review standards.
- **Engineering enablement and transformation leaders** — introducing AI-assisted development responsibly across teams.

Prerequisites: familiarity with software delivery, requirements, testing, or code review. Experience with Git, repositories, Markdown, APIs, or an AI coding tool is helpful. Participants do not need to be experts in every tool covered; the program focuses on transferable SDD patterns and practical workflow design.

Course Outline

Day 1 — From Vibe Coding to Context-Rich, Executable Specifications

Understand why AI delivery fails without explicit intent and context, prepare a codebase for specification work, and turn business notes into a hardened spec, plan, and task list.

Topic · From Vibe Coding to Spec-Driven AI Delivery & the SDD Tool Landscape

Participants understand why specs are now the control layer for AI delivery, and which tools support which part of the lifecycle

- Why AI coding fails without specs; hallucination patterns and context loss
- Difference between prompt, spec, plan, task, and test; the new role of BA/Dev/QA/PM
- GitHub Spec Kit, Kiro, Claude Code, Codex, Cursor, Copilot, Aider/OpenCode — where each tool fits
- Spec-first versus chat-first versus IDE-agent model

Hands On — Give the same vague requirement to an AI tool, observe the output, then improve it using a structured spec; map your team's SDLC to best-fit SDD tooling

Topic · Repository Context Engineering & Codebase Understanding

Participants learn to prepare clean context and extract system understanding before asking AI to generate specs

- Why specs must be grounded in existing systems; context discovery and repository packaging
- Token budget and deliberate file selection

- Knowledge graph for code: module relationships, call flows, dependency tracing, and architecture extraction

Hands On — Use Repomix/code2prompt to create AI-ready repo context, then use graph-style analysis to identify modules, dependencies, flows, and spec boundaries

Topic · AI-Ready Functional Specs & Clarification Loops

Participants generate business specs precise enough for AI agents, and use one model to generate while another challenges

- Problem statement, persona, workflow, and business rules
- Edge cases, constraints, assumptions, out-of-scope decisions, and acceptance criteria
- AI-assisted requirement interrogation; missing rules and conflict detection
- Decision tables, examples, counterexamples, and Given-When-Then scenarios

Hands On — Generate a functional spec from business notes, manually harden it, then run a “spec critic” prompt to detect ambiguity and generate clarification questions

Topic · GitHub Spec Kit Workflow

Participants understand how modern SDD tools convert specs into buildable tasks

- /specify, /plan, /tasks, and implementation flow
- Spec as source of truth; supported agents; repository-level workflow

Hands On — Create a feature using a Spec Kit-style flow: spec ' plan ' tasks

Day 1 Labs — Vague-requirement comparison · SDD tool mapping · repository context packaging · codebase graphing · functional-spec generation and hardening · ambiguity and clarification loop · Spec Kit workflow.

Day 2 — Agent Guardrails, Technical Contracts, Tests & Review

Turn functional intent into persistent agent guidance, technical specs, and machine-readable contracts; then test and review AI-generated code against the source of truth in a capstone.

Topic · Steering, Hooks & Agent Configuration Files

Participants create persistent, project-level guardrails for coding agents

- Kiro-style requirements ' design ' tasks; steering files, hooks, and project memory
- AGENTS.md, CLAUDE.md, Cursor Rules, and Copilot custom instructions; a project constitution
- What to put — and what not to put — in persistent instructions

Hands On — Build an agent instruction pack for a sample enterprise repo: product, tech, structure, and coding rules

Topic · Technical Specs & Machine-Readable Contracts

Developers produce technical specs and contracts that reduce wrong-code generation

- Architecture context, folder structure, API patterns, database rules, naming, logging, and error handling
- Security, performance, and testing rules
- OpenAPI, AsyncAPI, JSON Schema, DBML, Mermaid sequence/ER diagrams, and state diagrams — why AI agents need contracts

Hands On — Generate a technical spec from an existing repo and desired feature, then produce OpenAPI + JSON Schema + a Mermaid sequence diagram for it

Topic · Specifying AI Features and LLM Behaviors

Participants learn how to specify AI features, not just normal software features

- Prompt spec versus product spec; model behavior and input/output contract
- Grounding and RAG constraints; fallback, refusal behavior, confidence, and human escalation

Hands On — Write an AI feature spec for a document Q&A, support, or claims assistant

Topic · Test Generation and Code Review Against Specs

QA and developers turn specs into executable validation assets and audit AI-generated output instead of trusting it blindly

- Acceptance tests and Gherkin; unit, API, UI, model-behavior, and hallucination tests; golden datasets
- Spec-code traceability and an AI output review checklist
- Architecture violation detection, security and diff review, and regression risk

Hands On — Convert acceptance criteria into Gherkin and Postman/Playwright-style tests; use AI as a reviewer to compare generated code against the spec

Topic · Capstone: End-to-End Tool-Led SDD Lab

Participants leave with a practical SDD playbook they can apply immediately

- Business brief ' repo context ' functional spec ' technical spec
- API/data spec ' tasks ' tests ' review checklist

Hands On — Team capstone: create a full AI-agent-ready delivery pack and optionally generate a small code change

Day 2 Labs — Steering and agent instruction pack · technical-spec generation · API and data contracts · AI feature behavior specification · acceptance-test generation · AI-generated code review and traceability audit · end-to-end SDD delivery pack.

Capstone — End-to-End Tool-Led SDD Lab

Participants work in teams to take a business brief through a complete specification-driven delivery workflow:

- Prepare repository context with deliberate file selection
- Produce a Graphify/Repomix system summary
- Generate and harden a functional specification
- Challenge ambiguity and resolve missing decisions
- Create a technical specification grounded in the existing codebase
- Define API, data, workflow, or AI behavior contracts
- Convert the specification into implementation tasks
- Generate acceptance, unit, API, UI, or model-behavior tests
- Review an implementation or small code change against the source of truth
- Produce a delivery pack that another engineer and an AI coding agent can use

Acceptance is based on traceability: the team must show how the business brief becomes a spec, how the spec becomes tasks and tests, and how implementation is checked against the decisions captured in the spec.

What participants take away

- A practical explanation of why specs are the control layer for AI delivery
- An SDD tool-selection worksheet
- An AI-ready repository context package
- A codebase graph and architecture-discovery approach
- A hardened functional specification
- A repeatable clarification and ambiguity-detection loop
- A Spec Kit-style specification-to-plan-to-task workflow
- Steering documents, hooks, and project memory patterns
- An `AGENTS.md` / `CLAUDE.md`-style agent instruction pack
- A technical specification grounded in an existing repository
- Machine-readable API, data, workflow, and AI behavior contracts
- A test-generation workflow from acceptance criteria
- A spec-code traceability and AI-generated code review checklist
- A complete AI-agent-ready delivery pack
- An end-to-end SDD playbook ready for immediate use

A note on delivery

This is a build-and-review program, not a prompt-writing seminar or a tour of AI coding tools. Each day combines focused instruction, guided demonstrations, tool comparisons, repository analysis, specification workshops, test generation, code review, troubleshooting, and team delivery work.

The through-line is a single delivery workflow that evolves from an ambiguous business brief into repository context, functional and technical specifications, machine-readable contracts, implementation tasks, executable tests, and reviewable code.

Participants leave with a shared language for AI-assisted delivery and reusable artifacts—not just a collection of prompts or a demo that works once.