Faysal Aziz

Kiro Agentic Engineering Accelerator

From Specification to Verified Software

FA-0277 · Three-Day

Agentic & Generative AI · Software Development

Introduction

Three-Day Hands-On Course in Spec-Driven Development and Harness Engineering

AI coding tools can produce code quickly. Producing software that is correct, maintainable, traceable and safe to change is a much harder problem.

This intensive programme introduces a disciplined engineering approach built around **Kiro, spec-driven development and harness engineering**. Participants will learn to replace vague prompting and uncontrolled "vibe coding" with executable specifications, structured project context, automated controls and repeatable verification loops.

The programme is positioned as one of the **first trainer-led courses in Malaysia to bring spec-driven development and harness engineering together as a practical software-engineering discipline**. It is led by an instructor with **more than 30 years of industry experience**, using real industry-demanded practices rather than academic theory detached from delivery realities.

This is a build-first course. Participants will work inside Kiro, create specifications, configure steering, implement agent hooks, construct verification controls and complete an integrated capstone system. Lectures are kept deliberately short. Most of the programme is spent designing, building, testing, breaking and improving working software.

Kiro currently structures agentic development around specifications, steering and event-driven hooks. Its specification workflow converts requirements into design and implementation tasks, while harness engineering adds the surrounding controls, tools, feedback loops and verification needed to make agent-generated work dependable.

Learning outcomes

Upon successful completion, participants will be able to:

- Configure Kiro for structured, repository-aware development
- Distinguish prompting, context engineering, spec-driven development and harness engineering
- Convert incomplete business requests into clear, testable requirements
- Develop Kiro requirements, design and task specifications
- Establish persistent project context through steering documents
- Control agent behaviour using instructions, boundaries and repository conventions
- Configure hooks for automated quality and workflow enforcement
- Build verification loops around agent-generated code
- Design practical coding-agent harnesses with tools, checks, permissions and feedback
- Diagnose specification drift, context failure and unreliable agent behaviour
- Maintain traceability between requirements, implementation and validation
- Deliver a working software feature through an integrated capstone project

Prerequisites

Participants must meet all of the following requirements:

- Professional software-development experience
- Working knowledge of at least one mainstream programming language
- Ability to read, modify and debug an existing codebase
- Practical Git and GitHub or equivalent source-control experience
- Familiarity with branches, commits, pull requests and merge conflicts
- Working knowledge of automated testing
- Basic command-line proficiency
- A laptop capable of running the selected development stack
- Administrator rights to install and configure development tools
- A working Kiro installation and active account before the course begins
- Access to Git, the required runtime, package manager and project dependencies
- Reliable internet access
- Willingness to work intensively on a continuous three-day project

This programme is **not suitable for complete programming beginners**. Participants who cannot independently clone, run, test and troubleshoot a software project may struggle to complete the labs and capstone.

Training outline

1. Agentic Engineering with Kiro

From AI Code Generation to Controlled Engineering

1. Limitations of prompt-only development
2. Vibe coding versus engineering discipline
3. Human responsibility in agent-assisted delivery
4. Agent legibility and repository legibility
5. Specification, implementation and verification boundaries

Kiro Architecture and Working Model

1. Kiro IDE workspace
2. Agent interaction modes
3. Repository awareness
4. Specs
5. Steering
6. Hooks
7. Tool access and execution controls

Practical Environment Preparation

1. Kiro installation validation
2. Project workspace configuration
3. Git repository preparation
4. Runtime and dependency verification
5. Test-command validation
6. Baseline project execution

2. Spec-Driven Development Fundamentals

Intent Before Implementation

1. Business intent
2. Functional boundaries

3. Assumptions and constraints
4. Acceptance conditions
5. Edge cases
6. Non-functional expectations

Kiro Specification Workflow

1. Specification workspace structure
2. Requirements document
3. Design document
4. Implementation task document
5. Specification review and approval flow
6. Requirement-to-task traceability

Engineering Effective Requirements

1. User-centred requirements
2. Testable acceptance criteria
3. Ambiguity removal
4. Requirement decomposition
5. Failure-path requirements
6. Security and operational requirements
7. Scope-control techniques

Translating Requirements into Design

1. Existing-system analysis
2. Component responsibilities
3. Interfaces and contracts
4. Data-flow decisions
5. Dependency boundaries
6. Error-handling strategy
7. Testing strategy
8. Design consistency checks

Building Executable Task Plans

1. Incremental task decomposition
2. Dependency ordering
3. Validation checkpoints
4. Requirement references
5. Implementation boundaries
6. Completion criteria

3. Hands-On Feature Delivery with Kiro Specs

Repository Exploration

1. Existing architecture discovery
2. Dependency identification
3. Convention detection
4. Test-suite discovery
5. Risk-area identification

Specification Construction

1. Feature brief conversion
2. Requirements refinement
3. Design generation
4. Task-plan generation
5. Human review gates
6. Specification correction

Controlled Implementation

1. Task-by-task execution
2. Change inspection
3. Test execution
4. Defect correction
5. Incremental commits
6. Specification alignment checks

Specification Lifecycle Management

1. Requirement changes
2. Design amendments
3. Task-plan regeneration
4. Implementation drift
5. Stale-spec detection
6. Specification maintenance

4. Steering and Persistent Project Context

Kiro Steering Fundamentals

1. Product context
2. Technical context
3. Project structure
4. Coding conventions
5. Domain terminology
6. Repository-specific instructions

Foundational Steering Files

1. Product steering
2. Technology steering
3. Structure steering
4. Naming and formatting policies
5. Testing expectations
6. Documentation expectations

Context Design

1. Global instructions
2. Conditional instructions
3. File-scoped context
4. Task-specific context
5. Context prioritisation
6. Context-conflict management

Steering Quality

1. Clear behavioural constraints
2. Relevant context selection
3. Instruction duplication
4. Conflicting guidance
5. Context overload
6. Steering validation

5. Harness Engineering for Coding Agents

Harness Engineering Principles

1. Model versus harness responsibilities
2. Context and control layers
3. Deterministic safeguards
4. Feedback loops
5. Verification boundaries
6. Human intervention points

Core Harness Components

1. Task specification
2. Context selection
3. Tool access
4. Project memory
5. Task state
6. Permissions
7. Observability
8. Verification
9. Failure attribution
10. Intervention records

Engineering Agent-Friendly Repositories

1. Discoverable project structure
2. Machine-readable conventions

3. Reliable development commands
4. Fast feedback mechanisms
5. Local validation scripts
6. Clear failure messages
7. Reproducible environments

Guardrails and Execution Boundaries

1. Command permissions
2. File-access boundaries
3. Protected resources
4. Destructive-operation controls
5. Secret-management boundaries
6. Human approval gates
7. Stop conditions

6. Kiro Hooks and Automated Enforcement

Agent Hook Fundamentals

1. Event-driven automation
2. File events
3. Prompt events
4. Agent lifecycle events
5. Manual triggers

Quality-Control Hooks

1. Automatic formatting
2. Static analysis
3. Unit-test execution
4. Type checking
5. Dependency checks
6. Documentation validation

Policy-Enforcement Hooks

1. Naming-rule enforcement
2. Architecture-boundary checks
3. Protected-file controls
4. Test-required policies
5. Secret-detection controls
6. Specification consistency checks

Hook Reliability

1. Deterministic commands
2. Failure handling
3. Actionable output
4. Execution cost
5. False-positive reduction
6. Hook maintenance

7. Verification and Feedback Loops

Trust Through Evidence

1. Code-generation risk
2. Test evidence
3. Requirement evidence
4. Behavioural verification
5. Completion evidence

Multi-Layer Verification

1. Formatting checks
2. Static analysis
3. Unit testing
4. Integration testing
5. Contract testing
6. Security validation
7. Acceptance validation

Agent Self-Correction

1. Failure capture
2. Diagnostic context
3. Controlled retry
4. Correction validation
5. Retry limits
6. Escalation conditions

Harness Observability

1. Agent-action visibility
2. Tool-call records
3. Test-result capture
4. Change summaries
5. Failure attribution
6. Audit trails

8. Capstone: Spec-to-Verified Software Delivery

Capstone Brief and Constraints

1. Business request analysis
2. Existing repository assessment
3. Scope and delivery boundaries
4. Mandatory quality gates
5. Definition of done

Specification Package

1. Requirements specification
2. Acceptance criteria
3. Technical design
4. Implementation task plan
5. Risk and constraint coverage

Project Harness

1. Steering documents
2. Repository instructions
3. Tool boundaries
4. Automated hooks
5. Validation commands
6. Permission controls
7. Human approval points

Agent-Assisted Implementation

1. Incremental task execution
2. Code and test generation
3. Controlled corrections
4. Specification updates
5. Version-control discipline

Verification and Hardening

1. Automated quality checks
2. Requirement traceability
3. Failure-path validation
4. Regression validation
5. Specification-drift review
6. Harness-effectiveness review

Capstone Completion

1. Working feature demonstration
2. Specification review
3. Harness review
4. Verification evidence
5. Repository handover
6. Technical retrospective

Course Positioning

Stop prompting for miracles. Start engineering reliable outcomes.

This programme is designed for practitioners who are ready to move beyond impressive demonstrations and build repeatable AI-assisted software-delivery systems. The standard is not whether Kiro can generate code. The standard is whether the resulting change can be understood, tested, verified, maintained and trusted.

Disclaimer

This course outline is provided as a professional training guideline and reflects the intended scope at the time of preparation. The trainer reserves the right to amend, reorganise, substitute or omit topics, tools and activities where reasonably necessary to accommodate participant capability, software changes, environmental constraints, operational considerations or emerging industry practices. Such adjustments may be made without prior notice while preserving the programme's principal learning objectives.

© 2026 Faysal Aziz · Kuala Lumpur