Faysal Aziz

From Traditional Code to Agentic Systems

A Three-Day Architectural Briefing for Experienced Developers

FA-0216 · Three-Day

Agentic & Generative AI · AI for Leaders & Business · Software Development

---

Introduction

Understand the models. Control the process. Engineer the system around the AI.

Artificial intelligence changes software development in a way that ordinary framework upgrades do not. A developer is no longer working only with systems that follow explicit instructions and return predictable results. Large language models interpret incomplete information, generate probable answers, use external tools, and sometimes produce convincing results that are incorrect. For experienced developers, the challenge is therefore not learning another programming syntax. It is learning how to design dependable software around a component that is powerful, useful, and fundamentally non-deterministic.

This three-day course builds that understanding from the ground up. It begins with the mechanics of modern AI: tokens, embeddings, context windows, retrieval, prompts, tools, and agent loops. It then moves into spec-driven development, where specifications become central engineering artifacts rather than documents that are forgotten once coding begins. GitHub's current Spec Kit process, for example, formalizes a flow from project principles and specifications through clarification, planning, tasks, analysis, implementation, and convergence.

The course also connects specifications to the broader discipline of harness engineering. The model alone is not the application. The surrounding harness supplies context, instructions, tools, permissions, execution controls, validation, tracing, and feedback. Current industry guidance increasingly treats this surrounding system as the mechanism that turns raw model capability into controlled software behaviour.

The final part examines multi-agent orchestration and asynchronous architectures. Participants will explore why adding more agents does not automatically improve a system, how agents can review or verify one another, and how event-driven platforms such as Kafka can separate long-running tasks, maintain message history, and coordinate independent services. Event-driven agent architectures are increasingly being explored for scalable, loosely coupled multi-agent systems.

The course is entirely conceptual and architecture-focused. Instructor-led demonstrations will make the ideas concrete, but participants will not be required to complete hands-on labs. The instructor brings more than 30 years of industry experience and will present material based on real development pressures, operational risks, and current industry demand rather than an academic treatment of AI.

Learning outcomes

By the end of the course, participants will be able to:

- Explain how generative AI and large language models produce responses
- Describe tokens, embeddings, context windows, retrieval, RAG, and tool use
- Recognize the limitations, risks, and non-deterministic behaviour of AI systems
- Distinguish prompt engineering from context engineering
- Explain what makes an AI system an agent
- Identify where MCP and external tools fit into an agent architecture
- Structure clear specifications for AI-assisted software development
- Relate spec-driven development to harness engineering and the wider delivery lifecycle
- Position MVP scope within specifications, architecture, and controlled delivery
- Describe single-agent, multi-agent, and event-driven orchestration patterns
- Plan validation, observability, security, and human approval controls
- Evaluate when agentic architecture is useful and when conventional software is safer

Prerequisites

- Professional software development experience
- Familiarity with application architecture and APIs
- Basic understanding of databases and distributed systems
- General awareness of cloud services and message-based integration
- No previous AI, machine-learning, or data-science experience required
- No specialised mathematics required

Training outline

1. AI Foundations for Experienced Developers

What Artificial Intelligence Actually Is

1. AI, machine learning and generative AI
2. Training versus inference
3. Probability-based output generation
4. Deterministic software versus probabilistic systems
5. Common AI capabilities and misconceptions

Large Language Model Fundamentals

1. Model inputs and outputs
2. Tokens and tokenization
3. Context windows
4. Embeddings and semantic similarity
5. Attention and response generation
6. Model parameters and model size

Supplying Knowledge and Context

1. Prompt engineering
2. Role prompting
3. System, user and tool instructions
4. Context engineering
5. Context selection and ordering
6. Retrieval-Augmented Generation
7. Vector search and knowledge retrieval
8. Memory versus retrieved context

Models, Tools and External Systems

1. Function and tool calling
2. Model Context Protocol
3. APIs, databases and enterprise services

4. Structured outputs
5. Tool permissions and boundaries
6. Model, tool and application responsibilities

Understanding AI Agents

1. Model, instructions, tools and agent loop
2. Goals, decisions and actions
3. Role-based agent design
4. Workflows versus agents
5. Single-agent patterns
6. Autonomy levels
7. Human approval points

AI Limitations and Risks

1. Hallucinations
2. Non-deterministic behaviour
3. Knowledge and context limitations
4. Bias and unsafe output
5. Prompt injection
6. Data leakage
7. Cost, latency and reliability
8. Appropriate and inappropriate use cases

2. Spec-Driven Development and Harness Engineering

From Code-First to Spec-Driven Development

1. Purpose of spec-driven development
2. Specifications as primary engineering artifacts
3. Requirements versus implementation detail
4. Predictable development versus unstructured AI coding
5. Spec-driven development and traditional SDLC practices

Specification Structure

1. Problem statement
2. User and business outcomes
3. Scope and exclusions
4. Functional requirements
5. Non-functional requirements
6. Acceptance criteria
7. Constraints and assumptions
8. Dependencies and unresolved questions

Specification Workflow

1. Project principles and engineering rules
2. Specification creation
3. Requirement clarification
4. Technical planning
5. Task decomposition
6. Consistency checks
7. Implementation control
8. Review and convergence
9. Specification maintenance

MVP within Spec-Driven Delivery

1. MVP boundaries
2. Essential versus optional capabilities
3. Risk-first prioritisation
4. Vertical delivery slices
5. Measurable completion criteria
6. Controlled expansion beyond MVP

Understanding Harness Engineering

1. Model capability versus system capability
2. Instructions and context assembly
3. Tool registration and execution

4. Permissions and sandboxing
5. State and memory management
6. Validation and quality gates
7. Retries and recovery
8. Logging, tracing and evaluation

Connecting Specifications to the Harness

1. Specifications as agent context
2. Repository guidance and coding rules
3. Task-specific context
4. Automated checks
5. Test and review feedback
6. Progress tracking
7. Human decision points

Optimisation, Security and Governance

1. Context size and relevance
2. Model selection
3. Cost and latency controls
4. Secure tool access
5. Secrets and sensitive data
6. Least-privilege execution
7. Auditability and traceability
8. Architecture and security review gates

3. Multi-Agent Orchestration and Event-Driven AI Systems

Why Multi-Agent Systems Exist

1. Limits of a single agent
2. Role separation
3. Specialist agents
4. Parallel and sequential work
5. Coordination overhead
6. When multiple agents are unnecessary

Multi-Agent Design Patterns

1. Coordinator and worker
2. Planner and executor
3. Generator and reviewer
4. Agent-as-tool
5. Supervisor and specialist
6. Debate and consensus
7. Hierarchical orchestration

Reliability and Verification

1. Hallucination containment
2. Output validation
3. Agents verifying agents
4. Independent review
5. Confidence and evidence
6. Deterministic checks
7. Retry and fallback policies
8. Human-in-the-loop controls

Asynchronous Agent Architecture

1. Synchronous versus asynchronous execution
2. Events, commands and responses
3. Producers and consumers
4. Topics, queues and subscriptions
5. Kafka as an event backbone
6. Decoupled agent communication
7. Long-running tasks
8. Correlation and workflow state

Failure Management

1. Timeouts and partial failures
2. Duplicate and out-of-order messages
3. Idempotency

4. Retry limits
5. Dead-letter handling
6. Compensation and recovery
7. Escalation to human operators

Observability and the Black-Box Problem

1. Agent execution traces
2. Prompt and context records
3. Tool-call histories
4. Message lineage
5. Decision and approval logs
6. Metrics and operational alerts
7. Reproducibility limitations
8. Audit and incident investigation

Production Governance

1. Agent identity and permissions
2. Data access boundaries
3. Approval thresholds
4. Model and prompt versioning
5. Evaluation gates
6. Operational ownership
7. Safe deployment and rollback
8. Continuous monitoring

Disclaimer

This outline is intended as a structured guideline for a three-day, instructor-led programme. The trainer may adjust the sequence, emphasis, terminology, demonstration scope, or depth of individual topics according to participant experience, organisational priorities, available time, and relevant developments in the field. Such amendments may be made at the trainer's professional discretion without prior notice, while preserving the overall learning objectives of the programme.

© 2026 Faysal Aziz · Kuala Lumpur