Faysal Aziz

From AI Foundations to Agentic Applications

A two-day, developer-focused course on building dependable AI workflows, autonomous applications, adaptive interfaces, and production-ready engineering harnesses

FA-0213 · two-day

Agentic & Generative AI · Software Development · Data & Analytics

---

Introduction

Software development is moving beyond the simple use of AI chat interfaces. Developers are increasingly expected to understand how language models process information, how agents make decisions, how tools and external systems are connected, and how AI-generated changes can be evaluated before they reach production. This course addresses that shift without treating AI as magic or reducing the subject to a collection of prompting tricks.

The opening portion establishes the technical foundations required to work responsibly with modern AI systems. Participants will examine the relationship between artificial intelligence, machine learning, deep learning, transformers, and large language models. Particular attention will be given to tokens, context windows, probabilistic generation, embeddings, attention mechanisms, inference, and the practical reasons that models can produce inconsistent, incomplete, biased, or fabricated outputs. These concepts are necessary for developers who must decide where AI can be trusted, where deterministic controls are required, and where human approval must remain part of the workflow.

The course then moves from conventional prompt engineering into structured prompting for agentic systems. Participants will study how goals, constraints, context, tools, output contracts, validation criteria, and termination conditions can be expressed clearly enough for an agent to perform useful work. This is especially important because an agent is not simply a chatbot with a longer prompt. Agentic applications introduce planning loops, tool calls, state management, memory, retries, delegation, permissions, and operational risk.

Multi-agent orchestration will be addressed at an architectural level appropriate for a two-day programme. The course will distinguish between situations that benefit from specialised agents and situations where a single well-designed agent is safer and more efficient. It will cover supervisors, routers, workers, shared state, task handoffs, parallel execution, conflict resolution, observability, and cost controls without attempting to turn the programme into an advanced distributed-systems course.

The programme also introduces specification-driven development and **harness engineering**. In current agent-first engineering practice, the developer's responsibility increasingly includes preparing the specifications, repository structure, tools, constraints, feedback mechanisms, tests, and execution environment within which an AI coding agent operates. OpenAI describes harness engineering as work centred on the systems, scaffolding, abstractions, and controls that enable coding agents to complete meaningful engineering tasks reliably. (OpenAI) Participants will therefore learn to treat specifications, acceptance criteria, repository instructions, tests, linters, security gates, and review policies as part of the product's engineering harness rather than as optional documentation.

The applied portion is organised around the tools and outcomes identified for the programme: ChatGPT, Claude for development work, Gemini-based image generation, automated workflows, autonomous AI applications, Model Context Protocol integration, generative interfaces, adaptive user experiences, and pre-deployment code scanning. Claude Code, for example, is designed to inspect codebases, edit files, execute commands, and integrate with development tooling, while its subagent model supports specialised workers and tool access. Gemini's image capabilities support generation, editing, composition, and natural-language transformations, making them suitable for demonstrating multimodal application workflows rather than image creation as an isolated activity.

Model Context Protocol will be introduced as a standardised method for connecting AI applications to tools, data sources, prompts, and workflows. Its architecture separates hosts, clients, and servers and uses defined protocol primitives for exchanging context and invoking capabilities. Coverage will remain practical and appropriately scoped: participants will understand the architecture, configure or consume a simple integration, and examine security boundaries rather than attempting to build a large production MCP ecosystem within the available time.

The course is intended for senior and junior software developers, front-end developers, and QA automation engineers, with an anticipated cohort of approximately 23 participants. It is not a beginner programming course. Participants will be expected to work confidently with source code, APIs, command-line tools, version control, structured data, and modern application-development workflows.

The instructor has more than 30 years of industry experience and will deliver the programme using real, industry-demanded engineering practices rather than an academic or theory-heavy approach. The two-day schedule is deliberately selective: it provides enough foundation to understand agentic systems and enough guided implementation to build a coherent workflow, but it does not attempt to provide production mastery of every model, cloud platform, orchestration framework, or security discipline. AWS will receive only limited attention where a deployment or hosted service requires cloud context; the programme remains primarily platform-neutral.

Learning outcomes

Upon completing the course, participants should be able to:

- Explain the relationships among AI, machine learning, deep learning, transformers, and large language models.
- Describe tokenisation, context limitations, inference behaviour, attention, embeddings, and probabilistic generation.
- Identify hallucination, bias, prompt injection, data leakage, unsafe tool use, excessive autonomy, and other AI risks.
- Select an appropriate AI tool for coding, image generation, analysis, testing, or workflow automation.
- Develop structured prompts containing goals, context, constraints, output schemas, and acceptance criteria.
- Distinguish between conversational AI, tool-using assistants, autonomous agents, and multi-agent systems.
- Describe the principal components of an agent loop and multi-agent orchestration architecture.
- Apply specification-driven development and harness engineering principles to AI-assisted software delivery.
- Use AI development tools to inspect, generate, refactor, test, and review application code.
- Design a basic automated workflow that connects an AI model with application tools or services.
- Explain MCP hosts, clients, servers, tools, resources, prompts, permissions, and trust boundaries.
- Develop a small generative or adaptive user-interface workflow.
- Incorporate automated testing, linting, dependency checks, secret detection, and code scanning before deployment.
- Evaluate AI-generated work through deterministic checks and human review rather than relying on model confidence.
- Recognise where cloud deployment may be useful without making AWS the central focus of the solution.

Prerequisites

Participants must meet the following requirements:

- Current employment or practical experience in software development, front-end development, test automation, or a closely related engineering role.
- Confident programming ability in at least one mainstream language such as JavaScript, TypeScript, Python, Java, C#, or Go.
- Ability to read and modify an unfamiliar but reasonably structured codebase.
- Practical understanding of functions, modules, classes, exceptions, dependencies, and asynchronous operations.
- Working knowledge of HTTP, REST APIs, request and response structures, authentication concepts, and status codes.
- Ability to create, read, and validate JSON documents.

- Practical Git experience, including cloning, branching, committing, reviewing differences, and resolving basic merge conflicts.
- Competence using a terminal or command shell and installing project dependencies.
- Familiarity with an integrated development environment such as Visual Studio Code.
- Working knowledge of package-management and build tooling relevant to the participant’s chosen language.
- Basic experience with automated unit or integration testing.
- Basic understanding of environment variables, configuration files, secrets, and application logs.
- For front-end implementation, working knowledge of HTML, CSS, JavaScript, component-based development, and browser developer tools.
- A configured development laptop with Git, a supported programming runtime, an IDE, and permission to install approved development tools.
- Access to the organisation-approved versions of ChatGPT, Claude, Gemini, or equivalent services used during the training.
- Access to model APIs or approved training credentials where API-based activities are required.
- No introductory programming instruction, basic Git training, or remedial API development will be included.

Training outline

1. AI, Machine Learning and Large Language Model Foundations

Artificial Intelligence Terminology

1. Artificial intelligence
2. Machine learning
3. Deep learning
4. Generative AI
5. Foundation models
6. Large language models
7. Multimodal models

Deep Learning Fundamentals

1. Neural networks
2. Training data and parameters

3. Forward propagation
4. Loss functions
5. Backpropagation
6. Model training and inference
7. Transformer architecture
8. Attention mechanisms
9. Embeddings and vector representations

Tokens and Context Management

1. Tokenisation
2. Input and output tokens
3. Context windows
4. Context consumption
5. Conversation history
6. Retrieval and context injection
7. Token cost and latency considerations

How Language Models Generate Responses

1. Probabilistic generation
2. Next-token prediction
3. Temperature and sampling
4. Model reasoning behaviour
5. Deterministic and non-deterministic outputs
6. Model knowledge boundaries

AI Limitations

1. Hallucination
2. Incomplete context
3. Stale or missing knowledge
4. Mathematical and logical inconsistency
5. Ambiguous requirements
6. False confidence
7. Tool and data dependency
8. Model variability

AI Risk and Responsible Use

1. Bias and unfair outcomes
2. Privacy and confidential information
3. Intellectual-property considerations
4. Prompt injection
5. Insecure output handling
6. Excessive agency
7. Data and tool permissions
8. Human oversight
9. Auditability and accountability

2. Prompt Engineering for Agentic Systems

Prompting Fundamentals

1. Instructions
2. Context
3. Roles
4. Constraints
5. Examples and reference material
6. Output formatting
7. Evaluation criteria

Structured Prompt Design

1. Goal definition
2. Input contracts
3. Output schemas
4. Scope boundaries
5. Acceptance criteria
6. Error conditions
7. Completion conditions
8. Escalation conditions

Prompt Design for Software Engineering

1. Repository context
2. Technology constraints
3. Coding standards
4. Change boundaries
5. Testing requirements
6. Documentation requirements
7. Review requirements

Prompt Reliability

1. Instruction hierarchy
2. Context relevance
3. Prompt decomposition
4. Ambiguity reduction
5. Verification prompts
6. Structured outputs
7. Prompt versioning

3. Agentic AI Architecture

Agentic AI Concepts

1. Assistants versus agents
2. Goals and delegated tasks
3. Agent loops
4. Planning and execution
5. Observation and reflection
6. Tool use
7. State and memory
8. Termination controls

Agent Components

1. Model
2. System instructions

3. Tool registry
4. Working memory
5. Persistent memory
6. Planning mechanism
7. Execution environment
8. Validation layer
9. Human approval gates

Agent Control Patterns

1. Sequential workflows
2. Routing
3. Planning and execution
4. Reflection and correction
5. Retry and fallback
6. Human-in-the-loop
7. Bounded autonomy

Agent Safety and Governance

1. Least-privilege tool access
2. Sandboxed execution
3. Input validation
4. Output validation
5. Action approval
6. Timeout and iteration limits
7. Cost and token limits
8. Logging and traceability

4. Multi-Agent Orchestration

Multi-Agent Design

1. Specialised agents
2. Supervisor agents
3. Router agents
4. Worker agents

5. Reviewer agents
6. Shared and isolated context

Orchestration Patterns

1. Supervisor-worker
2. Sequential delegation
3. Parallel execution
4. Hierarchical orchestration
5. Debate and review
6. Handoff workflows

Coordination and State

1. Task decomposition
2. Agent responsibilities
3. Message passing
4. Shared state
5. Context boundaries
6. Result aggregation
7. Conflict resolution
8. Failure recovery

Multi-Agent Operational Concerns

1. Duplicate work
2. Context drift
3. Cascading errors
4. Tool contention
5. Latency
6. Token consumption
7. Observability
8. Single-agent alternatives

5. Specification-Driven Development and Harness

Engineering

Specification-Driven Development

1. Business requirements
2. Functional specifications
3. Technical constraints
4. Interface contracts
5. Acceptance criteria
6. Non-functional requirements
7. Testable completion conditions

Harness Engineering

1. Agent execution environment
2. Repository structure
3. Repository instructions
4. Tool configuration
5. Context preparation
6. Coding conventions
7. Architectural boundaries
8. Automated feedback loops

Engineering Controls for Coding Agents

1. Test suites
2. Linters and formatters
3. Type checking
4. Build validation
5. Dependency policies
6. Security policies
7. Review gates
8. Continuous integration checks

Agent-Friendly Software Repositories

1. Discoverable documentation

2. Modular architecture
3. Consistent naming
4. Machine-verifiable requirements
5. Reproducible environments
6. Clear ownership boundaries
7. Failure diagnostics

6. Practical Use of Current AI Tools

AI Tool Selection

1. ChatGPT
2. Claude
3. Gemini image generation
4. Model and tool capability comparison
5. Task suitability
6. Privacy and organisational approval
7. Cost and usage considerations

AI-Assisted Development

1. Codebase exploration
2. Requirement interpretation
3. Feature scaffolding
4. Code modification
5. Refactoring
6. Test generation
7. Defect analysis
8. Documentation generation
9. Pull-request review

Claude Development Workflows

1. Project context
2. Repository instructions
3. Tool permissions
4. File modification

5. Command execution
6. Subagents
7. Verification workflows

ChatGPT Development Workflows

1. Structured development requests
2. Code and architecture analysis
3. Tool-supported workflows
4. Iterative validation
5. Review and refinement

Gemini Image Workflows

1. Text-to-image generation
2. Image editing
3. Multi-image composition
4. Prompt-controlled transformations
5. Application asset generation
6. Multimodal input and output

7. Automated and Agentic Application Workflows

Workflow Design

1. Trigger
2. Inputs
3. Model interaction
4. Tool execution
5. Validation
6. Approval
7. Output delivery
8. Error handling

Static UI to Agentic Application Transition

1. Conventional user interfaces

2. AI-assisted interactions
3. Tool-enabled actions
4. Agent-managed workflows
5. User confirmation boundaries
6. Progress and status communication

Tool Integration

1. Function calling
2. API tools
3. File tools
4. Search and retrieval tools
5. Database access
6. Testing tools
7. Tool schemas
8. Tool-result validation

Workflow Reliability

1. Idempotency
2. Retry policies
3. Timeout management
4. Fallback behaviour
5. State persistence
6. Error classification
7. Operational logging

8. Model Context Protocol Integration

MCP Fundamentals

1. MCP hosts
2. MCP clients
3. MCP servers
4. Protocol lifecycle
5. Capability negotiation
6. JSON-RPC foundations

MCP Primitives

1. Tools
2. Resources
3. Prompts
4. Sampling
5. Notifications
6. Structured schemas

MCP Application Integration

1. Server discovery
2. Connection configuration
3. Tool exposure
4. Context retrieval
5. Tool invocation
6. Result handling
7. Client compatibility

MCP Security

1. Server trust
2. Authentication
3. Authorisation
4. Tool permissions
5. Data exposure
6. Input validation
7. User consent
8. Audit logging

9. Generative and Adaptive User Interfaces

Generative UI Concepts

1. Model-generated content
2. Structured UI definitions
3. Dynamic components

4. Tool-driven interface updates
5. Streaming responses

Adaptive Interface Behaviour

1. User intent
2. Context-aware presentation
3. Role-based adaptation
4. Progressive disclosure
5. Personalisation boundaries
6. Accessibility preservation

Front-End Integration

1. Model API integration
2. Structured response rendering
3. Loading and progress states
4. Error states
5. User confirmation
6. Output sanitisation
7. State synchronisation

10. Pre-Deployment Validation and Code Scanning

AI-Generated Code Review

1. Change inspection
2. Diff review
3. Architectural compliance
4. Logic validation
5. Test coverage
6. Human approval

Automated Quality Gates

1. Formatting
2. Linting

3. Type checking
4. Unit testing
5. Integration testing
6. Build validation

Security Scanning

1. Static application security testing
2. Dependency vulnerability scanning
3. Secret detection
4. Licence checks
5. Configuration scanning
6. Infrastructure scanning

Deployment Readiness

1. Environment configuration
2. Secrets management
3. Logging and monitoring
4. Rollback preparation
5. Release approval
6. Limited cloud deployment considerations
7. Platform-neutral deployment workflow

Disclaimer

This training outline is provided as an indicative programme framework and is intended to guide the scope, sequence, and anticipated coverage of the course. The trainer reserves the right to amend, reorder, expand, reduce, substitute, or omit any topic where reasonably necessary to accommodate participant readiness, technical constraints, tool availability, security requirements, organisational priorities, or developments in the relevant technologies. Such adjustments may be made at the trainer's professional discretion without prior notice, provided that the overall learning intent of the programme is substantially preserved.

© 2026 Faysal Aziz · Kuala Lumpur