Faysal Aziz

# AI-Assisted Software Development

## Build faster with AI while keeping developers (not models) in control

FA-0081 · three-day

Agentic & Generative AI · Software Development

---

## Introduction

AI coding tools can accelerate research, implementation, testing, refactoring, and documentation, but speed without engineering discipline creates a different kind of technical debt: plausible-looking defects, unnecessary rewrites, excessive token consumption, security exposure, and developers who repeatedly ask the model to debug problems it introduced.

This three-day course teaches software developers to use AI as an engineering accelerator rather than an unquestioned source of truth. Participants learn to define requirements before generating code, provide only the context required, constrain agent behaviour, validate every important output, and stop unproductive AI debugging loops. Python and C# are used for demonstrations and practical exercises in Visual Studio Code, with both IDE-based assistants and command-line agents such as Claude Code.

The curriculum incorporates the requested foundations, prompt engineering, specification-driven development, agentic workflows, security controls, and cost engineering. It also reflects current development environments in which coding agents can plan work, edit multiple files, execute commands, and iterate on failures—capabilities that make human review, permissions, context control, and acceptance criteria increasingly important.

The instructor brings more than 30 years of industry experience and will present practices drawn from real software delivery environments rather than treating AI-assisted development as a purely academic subject.

## Learning outcomes

Upon completion, participants will be able to:

- Explain how large language models generate code and why their output can appear correct while remaining technically wrong
- Select appropriate AI assistance for research, planning, implementation, testing, review, debugging, and documentation
- convert incomplete requirements into structured specifications and verifiable acceptance criteria
- Write concise engineering prompts that reduce ambiguity, unnecessary context, and repeated token consumption
- Apply a specification-to-plan-to-implementation workflow
- Use AI without surrendering architectural, security, or quality decisions
- Evaluate AI-generated Python and C# code through compilation, tests, static analysis, and manual review
- Prevent repeated AI debugging loops and recognise when conventional debugging is more efficient
- Use VS Code effectively for controlled AI-assisted development
- Operate CLI-based coding agents such as Claude Code with appropriate permissions and repository context
- Manage context windows, repository instructions, conversation scope, and task boundaries
- Apply human approval gates to agentic coding workflows
- Identify prompt injection, sensitive-data disclosure, insecure output handling, excessive agency, and tool-abuse risks
- Reduce AI development costs through model selection, context trimming, task decomposition, caching, and usage monitoring
- Establish practical team standards for reviewing and accepting AI-assisted code

# Prerequisites

- Working knowledge of software development
- Familiarity with either Python, C#, or another modern programming language
- Basic understanding of Git and source-control workflows
- Basic command-line experience
- Familiarity with software testing and debugging
- A development workstation with Visual Studio Code
- Python SDK and runtime
- .NET SDK
- Git client
- Access to an approved AI coding assistant
- Access to Claude Code or an equivalent CLI coding agent
- Permission to install the required VS Code extensions and command-line tools

- A non-production repository containing no confidential credentials or restricted organisational data

# Training outline

## 1. AI-Assisted Development as an Engineering Discipline

### The Role of AI in Modern Software Delivery

1. AI as an accelerator rather than an authority
2. Developer accountability for generated code
3. Appropriate and inappropriate uses of coding assistants
4. Productivity gains versus hidden engineering costs
5. Automation bias and overreliance
6. Maintaining developer knowledge and problem-solving ability

### AI, Machine Learning, Generative AI and Agents

1. Artificial intelligence
2. Machine learning
3. Generative AI
4. Large language models
5. Coding models
6. AI assistants
7. AI agents
8. Deterministic software versus probabilistic model output

### How Coding Models Produce Responses

1. Tokens and tokenisation
2. Context windows
3. Training data and learned patterns
4. Inference
5. Next-token prediction
6. Model parameters
7. Dense models
8. Mixture-of-experts models

9. Reasoning and thinking models
10. Quantisation
11. Model capability limitations
12. Knowledge cut-off limitations

### Why AI-Generated Code Fails

1. Hallucinated APIs
2. Incorrect library usage
3. Invalid assumptions
4. Missing edge cases
5. Version incompatibilities
6. Insecure defaults
7. Over-engineered solutions
8. Unnecessary dependencies
9. Silent behavioural defects
10. Plausible but incorrect explanations

## 2. Selecting the Right AI Interaction Mode

### Completion-Based Assistance

1. Line completion
2. Function completion
3. Repetitive code generation
4. Boilerplate generation
5. Localised refactoring

### Conversational Assistance

1. Technical research
2. Requirement clarification
3. Design exploration
4. Code explanation
5. Review support
6. Test-case identification

## Agentic Assistance

1. Multi-file changes
2. Repository exploration
3. Command execution
4. Test execution
5. Build execution
6. Iterative implementation
7. Autonomous correction loops

## Choosing the Lowest-Sufficient Level of Autonomy

1. Manual implementation
2. Inline completion
3. Chat-based generation
4. Edit mode
5. Agent mode
6. CLI agent
7. Human approval requirements
8. Task risk classification

# 3. Engineering Prompts for Software Development

## Prompt Anatomy

1. Role and operating context
2. Engineering objective
3. Current system state
4. Technical constraints
5. Scope boundaries
6. Input artifacts
7. Expected output
8. Acceptance criteria
9. Verification requirements

## Requirement Quality

1. Intent
2. Business purpose
3. Functional requirements
4. Non-functional requirements
5. Constraints
6. Assumptions
7. Dependencies
8. Exclusions
9. Definition of done

## Prompting Techniques

1. Zero-shot prompting
2. Few-shot prompting
3. Structured prompting
4. Constraint-based prompting
5. Schema-constrained output
6. Step-bounded instructions
7. Diff-oriented requests
8. Review-only requests
9. Plan-only requests
10. Test-first requests

## Reducing Prompt Ambiguity

1. Replacing vague verbs
2. Defining expected behaviour
3. Specifying supported versions
4. Defining error-handling expectations
5. Identifying performance requirements
6. Identifying security requirements
7. Establishing naming and style conventions

## Prompt Anti-Patterns

1. Asking the model to build an entire system at once
2. Supplying unrelated repository content
3. Mixing planning and implementation
4. Requesting unrestricted improvements
5. Accepting unspecified dependencies
6. Repeatedly submitting the same failing prompt
7. Asking the model to infer business-critical requirements
8. Treating generated explanations as verification

## Prompt Evaluation

1. Requirement coverage
2. Constraint compliance
3. Output relevance
4. Technical correctness
5. Context efficiency
6. Reusability
7. Verifiability

# 4. Specification-Driven AI Development

## From Intent to Engineering Specification

1. Problem statement
2. User and system context
3. Scope definition
4. Functional requirements
5. Non-functional requirements
6. Interface contracts
7. Data requirements
8. Error conditions
9. Security requirements
10. Operational requirements

## Requirement Decomposition

1. Epics and capabilities
2. Features
3. User stories
4. Engineering tasks
5. Technical dependencies
6. Implementation order
7. Risk areas

## Acceptance Criteria

1. Expected inputs
2. Expected outputs
3. Boundary conditions
4. Failure behaviour
5. Performance expectations
6. Security expectations
7. Observability expectations

## Development Artifacts

1. Product requirement document
2. Technical specification
3. Architecture decision record
4. Task breakdown
5. API contract
6. Data model
7. Test plan
8. Pull-request description

## AI-Assisted Research

1. Repository discovery
2. Existing implementation identification
3. Dependency investigation
4. Framework capability investigation

5. Version verification
6. Documentation verification
7. Source credibility
8. Separating facts from model inference

### Plan-Before-Code Workflow

1. Repository assessment
2. Change-impact analysis
3. Proposed file changes
4. Dependency changes
5. Test strategy
6. Rollback considerations
7. Human approval of the implementation plan

## 5. Controlled Implementation Workflow

### The Spec-to-Research-to-Plan-to-Implement Lifecycle

1. Intent confirmation
2. Context collection
3. Requirement decomposition
4. Technical research
5. Implementation planning
6. Incremental coding
7. Verification
8. Review
9. Acceptance

### Small-Batch Code Generation

1. Single-responsibility tasks
2. Bounded file changes
3. Minimal diffs
4. Incremental compilation
5. Incremental testing
6. Frequent checkpoints

## Human-in-the-Loop Controls

1. Plan approval
2. Dependency approval
3. File-change approval
4. Command-execution approval
5. Migration approval
6. Security-sensitive change approval
7. Final merge approval

## Preserving Existing System Behaviour

1. Regression awareness
2. Public interface stability
3. Backward compatibility
4. Configuration compatibility
5. Data compatibility
6. Logging compatibility
7. Deployment compatibility

## Preventing Unnecessary AI Rewrites

1. Targeted edits
2. Diff size limits
3. Existing-pattern reuse
4. Change isolation
5. Refactoring boundaries
6. Formatting-only change exclusion
7. Unrelated cleanup exclusion

# 6. AI-Assisted Python Development

## Python Project Context

1. Project structure
2. Python version
3. Dependency management

4. Virtual environments
5. Coding conventions
6. Type-hinting expectations

## Python Code Generation

1. Functions and classes
2. Data models
3. File and data processing
4. API integration
5. Exception handling
6. Asynchronous code

## Python Verification

1. Syntax validation
2. Type checking
3. Unit testing
4. Integration testing
5. Linting
6. Formatting
7. Dependency validation

## Python Code Review

1. Mutable default arguments
2. Exception scope
3. Resource handling
4. Input validation
5. Type consistency
6. Concurrency behaviour
7. Dependency risks
8. Security-sensitive operations

# 7. AI-Assisted C# and .NET Development

## .NET Project Context

1. Solution structure
2. Project structure
3. Target framework
4. NuGet dependencies
5. Nullable reference types
6. Coding conventions
7. Build configuration

## C# Code Generation

1. Classes and interfaces
2. Records and data-transfer objects
3. Dependency injection
4. Asynchronous programming
5. LINQ operations
6. Configuration handling
7. API endpoints
8. Exception handling

## .NET Verification

1. Compilation
2. Static analysis
3. Unit testing
4. Integration testing
5. Code formatting
6. Package validation
7. Framework compatibility

## C# Code Review

1. Async and await correctness
2. Cancellation-token handling

3. Nullability
4. Resource disposal
5. Thread safety
6. Dependency lifetimes
7. Exception propagation
8. Input validation
9. Security-sensitive APIs

# 8. Visual Studio Code AI Development Environment

## Workspace Preparation

1. Repository opening
2. Workspace trust
3. Language extensions
4. Python extension
5. C# Dev Kit
6. Git integration
7. Terminal integration
8. Testing integration

## AI Interaction Modes in VS Code

1. Inline completion
2. Chat
3. Edit workflows
4. Agent workflows
5. Terminal assistance
6. Repository-aware context

## Context Selection

1. Current file
2. Selected code
3. Related files
4. Symbols
5. Errors

6. Test output
7. Terminal output
8. Git changes

## Controlled Agent Execution

1. Reviewing proposed edits
2. Reviewing generated commands
3. Restricting workspace scope
4. Monitoring multi-file changes
5. Inspecting diffs
6. Reverting unwanted changes
7. Separating generated and accepted changes

## Repository Instructions

1. Architecture conventions
2. Build commands
3. Test commands
4. Coding standards
5. Dependency policies
6. Security restrictions
7. Prohibited operations
8. Definition of done

# 9. CLI-Based Coding Agents with Claude Code

## CLI Agent Fundamentals

1. Terminal-based interaction
2. Repository discovery
3. File reading and editing
4. Shell-command execution
5. Build and test execution
6. Iterative task completion

## Claude Code Project Preparation

1. Repository scope
2. Project instruction files
3. Approved commands
4. Build instructions
5. Test instructions
6. Style conventions
7. Architectural constraints
8. Security restrictions

## Effective CLI Agent Workflow

1. Explore before editing
2. Request a plan
3. Review the plan
4. Authorise bounded implementation
5. Inspect the diff
6. Execute deterministic checks
7. Correct specific failures
8. End completed sessions

## Permission and Tool Controls

1. Read permissions
2. Write permissions
3. Command permissions
4. Network access
5. External tool access
6. Destructive command restrictions
7. Approval prompts

## Context Management

1. Session scope
2. Context compaction
3. Clearing obsolete context

4. Referencing relevant files
5. Excluding generated artifacts
6. Separating unrelated tasks
7. Maintaining project instructions

### CLI Agent Anti-Patterns

1. Starting without repository instructions
2. Granting unrestricted command execution
3. Combining unrelated changes
4. Allowing uncontrolled dependency installation
5. Continuing after repeated failed attempts
6. Accepting unexplained file modifications
7. Using the agent as the only reviewer

## 10. Verification of AI-Generated Code

### Evidence-Based Acceptance

1. Compilation evidence
2. Test evidence
3. Static-analysis evidence
4. Runtime evidence
5. Requirement traceability
6. Manual review evidence

### Layered Verification

1. Syntax validation
2. Type validation
3. Unit tests
4. Integration tests
5. Contract tests
6. Security tests
7. Performance checks
8. Manual exploratory checks

## AI-Assisted Test Development

1. Test-case identification
2. Boundary-value analysis
3. Negative testing
4. Error-path testing
5. Regression testing
6. Test-data generation
7. Test adequacy review

## Reviewing Generated Tests

1. Tests that reproduce implementation logic
2. Missing assertions
3. Over-mocking
4. False-positive tests
5. Missing boundary cases
6. Non-deterministic tests
7. Tests disconnected from requirements

## Code Review with AI

1. Review against requirements
2. Review against architecture
3. Review for maintainability
4. Review for security
5. Review for performance
6. Review for compatibility
7. Independent human confirmation

# 11. Breaking the AI Debugging Loop

## Recognising Unproductive Iteration

1. Repeated speculative fixes
2. Alternating between previous solutions
3. Expanding change scope

4. Introducing unrelated refactoring
5. Repeated dependency changes
6. Rising token usage without new evidence
7. Loss of the original failure state

## Evidence-First Debugging

1. Reproducing the defect
2. Capturing the exact error
3. Identifying the smallest failing case
4. Inspecting logs and stack traces
5. Checking recent changes
6. Confirming runtime and dependency versions
7. Forming a testable hypothesis

## Bounded AI Debugging

1. Supplying the exact failure evidence
2. Requesting diagnosis before modification
3. Limiting the number of hypotheses
4. Limiting the files that may change
5. Requiring justification for each change
6. Running one experiment at a time

## Stop Conditions

1. Maximum correction attempts
2. No new diagnostic evidence
3. Expanding blast radius
4. Repeated regression
5. Unexplained model behaviour
6. Need for framework documentation
7. Need for human specialist review

## Switching to Conventional Debugging

1. Breakpoints

2. Step-through debugging
3. Logging
4. Profiling
5. Binary search through commits
6. Dependency isolation
7. Minimal reproduction
8. Manual documentation review

# 12. Context and Token Efficiency

## Token Economics

1. Input tokens
2. Output tokens
3. Cached tokens
4. Reasoning tokens
5. Context accumulation
6. Tool-execution overhead
7. Agent iteration costs

## Context Selection

1. Minimum necessary context
2. Relevant files only
3. Relevant symbols only
4. Focused error output
5. Concise requirements
6. Current implementation state

## Context Reduction

1. Removing repeated instructions
2. Summarising prior decisions
3. Excluding logs without diagnostic value
4. Excluding build artifacts
5. Excluding vendor and generated directories
6. Starting a new session for unrelated work

## Task Decomposition

1. Research tasks
2. Planning tasks
3. Implementation tasks
4. Testing tasks
5. Review tasks
6. Documentation tasks

## Model Selection

1. Completion model
2. General coding model
3. Reasoning model
4. Fast model
5. High-capability model
6. Escalation criteria

## Cost-Control Techniques

1. Plan before implementation
2. Reuse stable repository instructions
3. Prompt templates
4. Context caching
5. Output-length constraints
6. Tool-call limits
7. Retry limits
8. Session termination
9. Usage monitoring
10. Team spending controls

# 13. Agent Architecture for Developers

## Agent versus Workflow

1. Fixed workflow
2. Model-directed workflow

3. Tool-using agent
4. Autonomous coding agent

## Core Agent Components

1. Model
2. System instructions
3. Context
4. Tools
5. Memory
6. State
7. Control loop
8. Stop conditions

## Tool Calling

1. File tools
2. Search tools
3. Build tools
4. Test tools
5. Source-control tools
6. External-service tools

## State and Memory

1. Session state
2. Task state
3. Repository state
4. Persistent instructions
5. Short-term memory
6. Long-term knowledge

## Agent Loop Design

1. Observe
2. Plan
3. Act

4. Verify
5. Recover
6. Escalate
7. Terminate

### Reliability Controls

1. Maximum iterations
2. Time limits
3. Token limits
4. Tool limits
5. Approval gates
6. Idempotent operations
7. Retry policies
8. Failure recovery

### Single-Agent and Multi-Agent Patterns

1. Planner and implementer
2. Implementer and reviewer
3. Developer and test agent
4. Security-review agent
5. Context duplication risks
6. Coordination overhead
7. Cost multiplication
8. Conflicting recommendations

## 14. Agent Harness and Workflow Governance

### Agent Harness Components

1. Instruction layer
2. Context builder
3. Tool registry
4. Permission layer
5. Execution environment
6. State manager

7. Evaluation layer
8. Observability layer

## Controlled Context Injection

1. Repository instructions
2. Technical specifications
3. Coding standards
4. API contracts
5. Security policies
6. Relevant source files

## Tool Registry Design

1. Approved tools
2. Tool schemas
3. Parameter validation
4. Permission boundaries
5. Audit logging
6. Failure handling

## Evaluation Pipelines

1. Build evaluation
2. Test evaluation
3. Static-analysis evaluation
4. Security evaluation
5. Requirement evaluation
6. Human review

## Observability

1. Prompt logging
2. Tool-call logging
3. File-change logging
4. Token usage
5. Cost tracking

6. Failure tracking
7. Approval history

### Failure Recovery

1. Rollback
2. Checkpoint restoration
3. Retry with revised context
4. Human escalation
5. Session termination
6. Incident review

## 15. Enterprise Knowledge and Retrieval-Augmented Development

### Retrieval-Augmented Generation Fundamentals

1. Embeddings
2. Vector representations
3. Chunking
4. Semantic search
5. Retrieval
6. Context injection

### Software Engineering Knowledge Sources

1. Architecture documentation
2. Coding standards
3. API documentation
4. Runbooks
5. Historical decisions
6. Known-issue databases
7. Approved implementation patterns

### Knowledge Quality

1. Source authority

2. Version relevance
3. Document freshness
4. Contradictory guidance
5. Deprecated content
6. Access permissions

### Retrieval Risks

1. Irrelevant context
2. Outdated documentation
3. Poisoned content
4. Excessive retrieval
5. Cross-project leakage
6. Sensitive-information exposure

## 16. Secure AI-Assisted Software Development

### Trust Boundaries

1. Developer workstation
2. Source repository
3. AI service
4. CLI agent
5. External tools
6. Build environment
7. Deployment environment

### Data Classification

1. Public information
2. Internal information
3. Confidential information
4. Personal data
5. Source code
6. Credentials and secrets
7. Regulated information

## AI Security Threats

1. Prompt injection
2. Indirect prompt injection
3. Sensitive-information disclosure
4. Insecure output handling
5. Supply-chain risks
6. Data and model poisoning
7. Excessive agency
8. Tool abuse
9. Context poisoning
10. Overreliance

## Secure Prompt and Context Handling

1. Sensitive-data exclusion
2. Secret redaction
3. Repository access control
4. Context validation
5. Retrieved-content distrust
6. Instruction precedence

## Secure Agent Tooling

1. Least privilege
2. Command allowlists
3. Network restrictions
4. File-system boundaries
5. Environment isolation
6. Credential isolation
7. Destructive-action approval

## Generated Code Security Review

1. Input validation
2. Output encoding
3. Authentication

4. Authorisation
5. Secret handling
6. Injection prevention
7. Error handling
8. Logging safety
9. Dependency safety

### Governance and Auditability

1. Approved AI tools
2. Acceptable-use policy
3. Data-retention requirements
4. Prompt and action logs
5. Source attribution
6. Code ownership
7. Review responsibilities
8. Compliance evidence

## 17. Team Adoption and Operating Standards

### AI-Assisted Development Policy

1. Approved use cases
2. Restricted use cases
3. Prohibited data
4. Approved tools and models
5. Required verification
6. Human accountability

### Definition of Done for AI-Assisted Code

1. Requirement traceability
2. Successful build
3. Successful tests
4. Static-analysis completion
5. Security review
6. Dependency review

7. Human code review
8. Documentation update

## Pull-Request Expectations

1. AI involvement disclosure
2. Change summary
3. Requirement mapping
4. Verification evidence
5. Security considerations
6. Known limitations

## Team Metrics

1. Accepted versus rejected suggestions
2. Review effort
3. Defect escape rate
4. Rework rate
5. Token consumption
6. Cost per completed task
7. Time to validated completion

## Sustainable Developer Practices

1. Maintaining foundational coding skills
2. Reviewing before accepting
3. Learning from generated code
4. Avoiding dependency on one model
5. Preserving architectural knowledge
6. Sharing validated prompt patterns
7. Recording failed AI approaches

# 18. Integrated Practical Development Workflow

## Python Implementation Track

1. Requirement definition

2. Technical specification
3. AI-assisted planning
4. Incremental implementation
5. Unit-test generation
6. Manual code review
7. Failure diagnosis
8. Token-use review

## C# Implementation Track

1. Requirement definition
2. Technical specification
3. AI-assisted planning
4. Incremental implementation
5. Build validation
6. Unit-test generation
7. Manual code review
8. Failure diagnosis
9. Token-use review

## VS Code Agent Track

1. Repository context configuration
2. Plan generation
3. Controlled multi-file editing
4. Diff inspection
5. Test execution
6. Change acceptance

## Claude Code CLI Track

1. Repository instruction configuration
2. Task scoping
3. Permission control
4. Plan review
5. Bounded implementation
6. Command review

7. Context management
8. Cost review

### Final Engineering Review

1. Requirement compliance
2. Functional correctness
3. Security
4. Maintainability
5. Test adequacy
6. Context efficiency
7. Token efficiency
8. Human ownership

## Disclaimer

This course outline is provided as an indicative training framework and may be refined, reordered, expanded, reduced, substituted, or otherwise amended by the trainer to reflect participant experience, organisational priorities, technical dependencies, software availability, security requirements, time constraints, and developments in the relevant technologies. Such adjustments may be made at the trainer's professional discretion without prior notice, provided that the overall learning objectives of the programme remain substantially represented.

© 2026 Faysal Aziz · Kuala Lumpur