Cloud Contraptions
TRAINING & CONSULTING

To discuss this course and customizations:
Call: +1 434-509-6890 or Email: sales@cloudcontraptions.com

AI for Software Developers

Class Duration

35 hours of live training delivered over 5 days.

Student Prerequisites

- At least one year of professional software development experience in any language
- Familiarity with Git
- No prior AI tool experience required

Target Audience

Professional software developers, tech leads, and engineering managers who want a structured, end-to-end path from AI-curious to AI-proficient, independent of any single programming language or vendor. Ideal for mixed-stack teams, consultancies, and platform groups whose members work across several languages, and for organizations that want one shared curriculum before choosing tools. Teams standardized on a single stack should consider the per-language edition of this intensive instead: AI-Assisted Development with Python, Rust, JavaScript and TypeScript, or C#.

Description

This course teaches developers to operate the way modern AI-augmented engineering teams do: with agentic AI coding assistants doing real implementation work under the developer's direction. The five days follow a deliberate arc. Participants start with how frontier models generate code and the prompting and context engineering techniques that reliably steer them, then survey the 2026 landscape of IDE assistants and agentic tools (GitHub Copilot and its coding agent, Claude Code, Cursor, OpenAI Codex, Windsurf) before moving to the heart of the course: delegating multi-step coding tasks to agents, reviewing and steering agent output, and using AI across the lifecycle for design, refactoring, testing, debugging, and documentation. The final phase covers extending assistants with project context, custom instructions, and tool integrations such as the Model Context Protocol (MCP), then the security, licensing, and governance practices that make adoption sustainable. Hands-on work uses current industry tools against production-

style codebases, with equal emphasis on productivity gains and the code-review judgment that keeps quality high when an agent writes the first draft. Several topics here also have shorter dedicated courses that go deeper on a single slice: Prompting and Context Engineering for Software Engineers for prompt craft, Building with Coding Agents for a decision-oriented agent survey, and Model Context Protocol for Developers for building MCP integrations; this intensive weaves the essentials of each into one coherent path.

Learning Outcomes

- Explain how frontier LLMs generate code, and choose the right model and tool tier for a given engineering task.
- Apply prompting and context engineering techniques that reliably produce correct, idiomatic code in any language.
- Compare the 2026 assistant landscape, from IDE assistants to agentic and terminal tools, and select the right tool for a task, team, and budget.
- Delegate multi-step implementation tasks to coding agents: scoping, planning, supervising, and iterating to completion.
- Review and steer agent output with the code-review discipline that keeps quality high when an agent writes the first draft.
- Use AI across the software lifecycle: design exploration, implementation, refactoring, testing, debugging, and documentation.
- Ground assistants in project context with instruction files, prompt libraries, and encoded team conventions.
- Extend assistants with external tools and data through the Model Context Protocol (MCP).
- Recognize security, licensing, and governance risks in AI-assisted development and apply appropriate guardrails.
- Measure productivity impact honestly and design a sustainable AI adoption plan for a team.

Training Materials

Comprehensive courseware is distributed online at the start of class. All students receive a downloadable MP4 recording of the training.

To discuss this course and customizations:
Call: +1 434-509-6890 or Email: sales@cloudcontraptions.com

Software Requirements

Students need a laptop with Git and a modern IDE or editor, plus access to an agentic AI coding assistant such as GitHub Copilot, Claude Code, Cursor, OpenAI Codex, or Windsurf (a free tier is sufficient). Labs provide production-style codebases in several mainstream languages, so no language-specific toolchain setup is required in advance.

Training Topics

How Coding Assistants and Frontier Models Work

- How LLMs generate code: tokens, context windows, and training data
- The frontier model families for coding: Claude, GPT, and Gemini compared
- Capability tradeoffs: speed, cost, context size, and reasoning depth
- Where AI excels in engineering work and where it reliably fails
- Assistant, pair programmer, or junior engineer: setting expectations

Prompting and Context Engineering for Code

- Writing prompts that produce correct, idiomatic code
- Context strategies: what to include from a large codebase
- Few-shot examples, chain-of-thought, and iterative refinement
- Structured outputs and constraining generation
- Conversation hygiene: when to iterate and when to start fresh

The 2026 Tool Landscape

- IDE assistants: GitHub Copilot, Cursor, and Windsurf
- Agentic and terminal tools: Claude Code, OpenAI Codex, and the Copilot coding agent
- Autonomy levels: completion, edit, multi-step, and cloud agents
- Selection criteria: team fit, platform commitments, and cost
- Multi-tool strategies and switching costs

Delegating Work to Coding Agents

- The agentic loop: plan, act, observe, iterate
- Writing task briefs agents can execute: scope, constraints, acceptance criteria
- Plan-then-execute sessions and the supervision spectrum
- Issue-to-PR workflows with cloud agents

- Running tests in the loop and iterating until green

Reviewing and Steering Agent Output

- A review checklist for AI-generated code
- Common failure modes: hallucinated APIs, subtle logic errors, over-generation
- Steering mid-task: interrupting, redirecting, and course-correcting
- Keeping diffs reviewable: small steps over heroic changes
- Maintaining authorial understanding of code you sign

AI Across the Lifecycle: Design, Implementation, and Refactoring

- Design exploration: architecture options, tradeoff analysis, and spikes
- Implementation workflows: from specification to reviewed code
- Multi-file refactors: planning, executing, and verifying
- Legacy modernization behind characterization tests
- Dependency and framework upgrades with AI assistance

AI-Driven Testing and Debugging

- Generating test suites: coverage gaps, edge cases, and fixtures
- Testing the tests: mutation-style review of generated suites
- Turning stack traces and logs into root-cause hypotheses
- Bisection and root-cause workflows with agents
- Knowing when the AI is guessing: validating proposed fixes

Documentation and Code Understanding

- Onboarding to unfamiliar codebases with AI guides
- Generating explanations, call graphs, and architecture summaries
- READMEs, how-to guides, and architecture decision records
- Summarizing modules, packages, and pull requests
- Keeping generated documentation honest and current

Project Context and Custom Instructions

- Instruction files across tools: CLAUDE.md, AGENTS.md, and rules files
- Encoding repo conventions: style, architecture, and testing norms
- Team prompt libraries: capturing and sharing what works
- Keeping context files current as the codebase evolves

Extending Assistants with MCP and Tool Integrations

- The Model Context Protocol: what it standardizes and why it matters

To discuss this course and customizations:
Call: +1 434-509-6890 or Email: sales@cloudcontraptions.com

- Connecting assistants to internal tools, databases, and documentation
- Evaluating, installing, and configuring existing MCP servers
- Security boundaries for tool-connected assistants
- When to build a custom integration vs. adopt an existing one

Security, Licensing, and Governance

- Prompt injection and untrusted content in agent workflows
- Secret handling and data boundaries: what never enters a prompt
- Common vulnerabilities in generated code
- Licensing and IP status of AI-generated code
- Organizational policy: approved tools, audit trails, and review requirements

Measuring Impact and Sustainable Adoption

- Metrics that matter: cycle time, review iteration, defect rate
- Vanity metrics and adoption anti-patterns
- Cost management: tokens, model tiers, and budgets
- Rollout patterns that work: champions, guilds, and paired onboarding
- Re-evaluating the tool landscape as it evolves

