APPLIED AND GENERATIVE AI

AI-Assisted Refactoring and Code Quality

Level: Practitioner • 2 days (expandable to 3) • Virtual, In-person

Overview

AI makes it easy to change a lot of code quickly. That is exactly why it can be dangerous: speed without discipline turns a tidy codebase into a subtly broken one. This course is about the opposite outcome, using AI to make code measurably better while keeping it working the whole way.

This is a hands-on, practitioner course. Rather than march through a catalog of every refactoring, it goes deep on the loop that keeps change safe and teaches you to apply it with an AI assistant doing the heavy lifting. We start from first principles: what refactoring actually is, what code quality means, and the safety net that has to be in place before you change anything. Only then do we let the AI loose, first on small, verifiable refactorings, then on the hard case everyone actually has, untested legacy code. Throughout, the emphasis is on judgment: reviewing what the AI produces and catching the subtle behavior changes it can introduce. Every module builds on the one before and ends with hands-on practice.

Who Should Attend

- Developers who want to improve existing code, not just write new code, with AI help
- Engineers who maintain legacy or unfamiliar codebases
- Technical leads raising code-quality standards across a team

  This is not an introduction to AI coding tools. Learners who have never used one should take *Foundations of AI Coding Assistants* first.

Prerequisites

- Comfortable using an AI coding assistant for everyday tasks
- Working proficiency in at least one programming language
- Basic familiarity with automated tests and version control

What You Will Learn

- Define what refactoring is and what code quality means, and judge both in real code
- Put a safety net of tests and version control in place before changing anything
- Use AI to understand unfamiliar and legacy code before touching it
- Perform small, safe refactorings in a tight change-and-verify loop
- Bring untested legacy code under test and refactor it incrementally
- Review AI-produced changes critically and catch subtle behavior changes

Course Outline

Day one: foundations and safe change

- Refactoring and Code Quality, Reframed for AI
  - What refactoring is: behavior-preserving change, and what it is not

  - What we mean by code quality: readability, simplicity, coupling, and tests
  - How AI changes the economics of refactoring, and what it does not change
  - Lab: assess a piece of code and name its quality problems
- The Safety Net: Tests and Version Control
  - Why you never refactor without a safety net, doubly so with AI
  - Using AI to strengthen test coverage before you change anything
  - Small commits and reversibility as guardrails
  - Lab: use AI to add tests around code you are about to change
- Understanding Before Changing
  - Using AI to read and explain unfamiliar or legacy code
  - Mapping dependencies and risk before touching anything
  - Lab: have the AI explain a legacy module, then verify the explanation

Day two: from small refactors to legacy code

- Small, Safe AI-Assisted Refactorings
  - The change-and-verify loop: one small step at a time
  - Common refactorings with AI: rename, extract, simplify, remove duplication
  - Keeping the AI on a short leash and reviewing every step
  - Lab: perform a series of small refactorings, verifying after each
- Refactoring Legacy Code with AI
  - Characterization tests: pinning down behavior you do not fully understand
  - Finding seams and breaking dependencies incrementally
  - Larger structural change without a big-bang rewrite
  - Lab: bring untested legacy code under test, then refactor a seam
- Reviewing AI's Work and Raising Quality
  - Reading AI output critically and spotting subtle behavior changes
  - Improving readability and naming, reducing complexity and duplication
  - Recognizing when an AI suggestion makes things worse
  - Lab: review and correct an AI-produced refactor, and judge the quality change

Extended Version

The three-day version keeps the same gradient and adds room to work at larger scale:

- Folding refactoring into everyday flow and pull requests
- Larger, multi-file refactoring campaigns with AI
- Simple code-quality metrics and tracking improvement over time
- A capstone that takes one real legacy component from tangled and untested to clean and covered