SOFTWARE DEVELOPMENT AND ENGINEERING PRACTICES

Clean Code and Code Reviews

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

Overview

Code is read far more often than it is written, and the cost of unclear code compounds quietly: every future change takes a little longer, every new team member ramps a little slower, every bug hides a little better. The hard part is not knowing that clean code matters; it is having concrete, defensible standards for what "clean" means, and a team process, the code review, that applies those standards without turning into nitpicking or rubber-stamping.

This is a hands-on, practitioner course. It builds the skill in the right order: first a working definition of clean code you can defend, then the craft of naming, functions, and structure, then refactoring as the safe way to get from the code you have to the code you want. Only then does it turn to code reviews, because a reviewer is applying exactly those judgments to someone else's work. Rather than catalog every rule ever written, the course goes deep on the judgments that matter most, practiced on real code. Every module ends with a lab and builds on the one before.

Who Should Attend

- Developers who want their code to be easier to read, change, and trust
- Technical leads responsible for code quality and the review process on their team
- Teams that want a shared, explicit standard instead of arguments about taste

Prerequisites

- Working proficiency in at least one programming language
- Some experience working in a shared codebase with other developers
- No prior code review experience required

What You Will Learn

- Explain what makes code clean in terms of cost of change, not taste
- Apply concrete standards for naming, functions, and code structure
- Recognize common code smells and judge which ones are worth fixing now
- Refactor safely in small steps, with tests as the safety net
- Give review feedback that improves the code and keeps the author on your side
- Design a team review process with clear standards, sensible scope, and fast turnaround

Course Outline

Day one: writing code people can read

- What Clean Means and Why It Pays
  - Reading versus writing: where the time actually goes
  - Cost of change as the honest measure of quality
  - The judgment behind the rules: context, tradeoffs, and knowing when to stop polishing

  - Lab: rank several versions of the same working code and defend your ranking
- Names, Functions, and Structure
  - Names that carry meaning and survive change
  - Small functions, single responsibility, and honest signatures
  - Structuring modules and classes so the design is visible in the code
  - Lab: rewrite a poorly named, tangled module into something a stranger could follow
- Code Smells and Refactoring
  - A working catalog of the smells that matter: duplication, long methods, deep nesting, leaky abstractions
  - Refactoring in small, safe steps, and why tests make courage cheap
  - Comments: the few that help, and the many that hide problems
  - Lab: identify the smells in a legacy module and refactor it incrementally under test

Day two: raising quality as a team

- The Code Review That Actually Works
  - What review is for: defects, design, knowledge sharing, and shared standards
  - What to look for first: correctness and design before style
  - Review scope and size: why small changes get better reviews
  - Lab: review a realistic pull request and compare what different reviewers caught
- Giving and Receiving Feedback
  - Comments that critique the code, not the coder
  - Distinguishing "must fix" from "consider" from personal preference
  - Being reviewable: writing pull requests that are easy to review well
  - Lab: rewrite a set of harsh or vague review comments into feedback that would land
- Making It Stick on a Team
  - Team standards: style guides, linters, and automating what machines do better
  - Review process design: turnaround expectations, approvals, and avoiding bottlenecks
  - Where AI code assistants and AI review fit, and where human judgment stays essential
  - Lab: draft a one-page code review standard for your own team

Extended Version

The three-day version keeps the same gradient and adds depth and sustained practice:

- Deeper refactoring practice on a larger legacy codebase, including characterization tests
- Design-level review: spotting architectural problems a diff view hides
- Metrics and health: measuring review effectiveness without gaming it
- A capstone: teams take a messy real-world module through refactoring, peer review, and a final quality walkthrough