SOFTWARE DEVELOPMENT AND ENGINEERING PRACTICES

Test-Driven Development and Automated Testing

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

Overview

Automated tests are what let a team change software quickly without fear, and their absence is why so many codebases calcify: nobody dares touch anything. Yet many teams that do write tests get little back, because their tests are slow, brittle, and aimed at the wrong level. Knowing how to write a test is the easy part. Knowing what to test, at which level, and how to keep the suite fast and trustworthy is the skill this course teaches.

This is a hands-on, practitioner course. It builds in dependency order: first the judgment layer, what tests are for and where each kind belongs, then the craft of writing good unit tests, then test-driven development as a design discipline, and only then the harder levels: test doubles, integration tests, and UI automation, where costs rise and discipline matters most. Rather than survey every tool, it goes deep on the practices that hold up in real codebases, grounded in tools like NUnit, Jest, Selenium, and Cypress. Every module ends with a lab and builds on the one before.

Who Should Attend

- Developers who want tests that pay for themselves instead of slowing them down
- Teams adopting TDD or trying to recover a slow, brittle test suite
- QA engineers moving from manual testing into test automation

Prerequisites

- Working proficiency in at least one programming language (labs support C# and JavaScript)
- Experience working in a shared codebase
- No prior automated testing experience required

What You Will Learn

- Judge what to test at which level using the test pyramid, and defend the tradeoffs
- Write unit tests that are fast, isolated, readable, and trustworthy
- Practice red-green-refactor TDD and use it to drive better design
- Isolate code under test with test doubles, and mock without creating brittleness
- Build integration and API tests that catch what unit tests cannot
- Automate UI tests with tools like Selenium and Cypress without building a flaky suite, and run everything in CI

Course Outline

Day one: the foundations, and test-driven development

- Why We Test, and What to Test Where
  - What automated tests actually buy: speed of change, not just bug-catching
  - The test pyramid: unit, integration, and UI, and the economics of each level

  - Qualities of a good test: fast, isolated, repeatable, and readable
  - Lab: assess a real codebase's test suite and identify what is tested at the wrong level
- Unit Testing Well
  - Anatomy of a unit test: arrange, act, assert
  - Naming, one behavior per test, and tests as documentation
  - Frameworks in practice: NUnit and xUnit for .NET, Jest for JavaScript
  - Lab: write a thorough unit test suite for an untested module and find the bug hiding in it
- Test-Driven Development
  - Red, green, refactor: the loop and the discipline behind it
  - How TDD drives design: testable code is decoupled code
  - When TDD earns its cost, and honest cases where it does not
  - Lab: build a feature strictly test-first, one red-green-refactor cycle at a time

Day two: the harder levels

- Test Doubles and Isolation
  - Stubs, mocks, fakes, and spies: what each is for
  - Mocking frameworks, and the over-mocking trap that makes suites brittle
  - Designing seams: dependency injection as testing's best friend
  - Lab: bring hard-to-test code under test by introducing seams and the right doubles
- Integration and API Testing
  - What integration tests must cover that unit tests cannot
  - Testing against real dependencies: databases and HTTP APIs
  - Managing test data and keeping integration suites fast enough to run always
  - Lab: write integration tests for an API, including its database behavior
- UI Automation and Continuous Testing
  - End-to-end testing with Selenium and Cypress: power, cost, and the flakiness problem
  - Selectors, waits, and patterns that keep UI tests stable
  - Tests in CI: the pipeline as the gatekeeper, and what to run when
  - Lab: automate the critical user journey of a web app and run the full test suite in a CI pipeline

Extended Version

The three-day version keeps the same gradient and adds depth where suites earn or lose their keep:

- Testing legacy code: characterization tests and safely bringing untested code under control
- Deeper end-to-end practice with Cypress and Playwright, including test architecture for large suites
- Where AI-assisted test generation helps, and how to review what it produces
- A capstone: take an untested feature from characterization tests through TDD to a full pyramid, running in CI