

Test-Driven Development and Refactoring Training Course

28 February – 4 March 2027
Toronto

AGILE LEADERS
Training Center

AGILE LEADERS
Training Center

Test-Driven Development and Refactoring Training Course

Ref: 2102_253993 **Date:** 28 February – 4 March 2027 **Location:** Toronto **Fees:** 12000 Euro

Overview

Test-Driven Development and Refactoring Training Course is a one-week intermediate course for software developers, test engineers, technical leads, and agile delivery teams who leave with a TDD Practice Portfolio. Participants use Red-Green-Refactor cycles, unit test design, test doubles, test seams, legacy-code characterization, continuous integration, and team working agreements to guide software design through executable feedback. Agile Leaders Training Center delivers this course in test-driven development and refactoring practice.

Who Should Attend

- Software development teams responsible for implementing maintainable application behavior
- Test engineering teams responsible for fast unit-level feedback and regression protection
- Technical leadership teams responsible for coding practices and design quality
- Agile delivery teams responsible for integrating development and verification
- Application maintenance teams responsible for changing legacy code safely

The course assumes participants can write, run, and debug object-oriented code and leaves out programming-language fundamentals, general Scrum, broad testing theory, and tool-only test automation.

Departments and Industries

The course supports departments that develop and maintain software in technology, financial, healthcare, industrial, and public-service environments.

- Software engineering and application development
- Quality engineering and developer enablement
- Technology architecture and platform teams
- Financial technology and digital services
- Healthcare systems and industrial software

Learning Objectives

By the end of this course, participants will be able to:

- Apply the Red-Green-Refactor cycle to small behaviors.
- Build readable unit tests and test lists.
- Compare state and behavior verification choices.
- Use test seams, stubs, fakes, and mocks.
- Refactor new and legacy code under tests.
- Evaluate TDD workflow and continuous-integration feedback.

Course Agenda

Day 1: Establish the TDD Cycle

- Red-Green-Refactor Cycle Map
- Behavior Example and Test List Template
- Arrange-Act-Assert Unit Test Structure
- Small-Step Implementation Decision Log
- Test Failure Diagnostic Checklist

Day 2: Design Reliable Unit Tests

- Test Naming and Intent Specification Pattern
- Boundary and Equivalence Example Matrix
- Test Fixture and Data Builder Method
- Test Smell Detection Checklist
- Unit Test Readability Review Grid

Day 3: Isolate Dependencies and Collaborators

- Test Seam Identification Canvas
- Dummy, Fake, Stub, Spy, and Mock Selection Matrix
- State versus Behavior Verification Decision Guide
- Dependency Injection and Isolation Pattern
- Test Coupling and Brittleness Review

Day 4: Refactor and Integrate TDD

- Characterization Test Plan for Legacy Code
- Code Smell and Refactoring Opportunity Map
- Behavior-Preserving Refactoring Sequence
- Continuous Integration Test Feedback Pipeline
- Team TDD Working Agreement and Review Cadence

AGILE LEADERS
Training Center

Day 5: Practice TDD Decisions

- Suggested Exercise: Build a Test List from a Behavior Example
- Suggested Exercise: Complete a Red-Green-Refactor Cycle
- Suggested Exercise: Select and Configure Test Doubles
- Suggested Exercise: Characterize and Refactor Legacy Code
- Capstone Exercise: Complete a TDD Practice Portfolio

Practical Exercises

The course uses suggested activities with fictional service, payment, scheduling, and device-control code examples.

- Suggested activity: convert payment rules into a sequenced unit test list.
- Suggested activity: isolate a healthcare scheduling dependency with a suitable test double.
- Suggested activity: characterize industrial device-control code before refactoring.
- Suggested activity: diagnose a brittle test suite and revise the team agreement.

FAQs

Who suits test-driven development training, and what does it assume?

Test-driven development training suits developers, test engineers, technical leads, and agile delivery teams. It assumes participants can write, run, and debug object-oriented code in their working language.

How does test-driven development differ from general software testing training?

Test-driven development uses tests to guide incremental software design before production code is written, while general software testing training covers broader verification levels, techniques, and quality activities after or alongside implementation.

What is the Red-Green-Refactor cycle in TDD?

Red-Green-Refactor starts with a small failing test, adds the minimum code needed to pass, and then improves code and test structure while preserving behavior. The cycle repeats for the next behavior.

When should TDD use mocks, stubs, or fakes?

TDD uses test doubles when a dependency makes a unit test slow, nondeterministic, hard to control, or difficult to observe. Teams select stubs, fakes, spies, or mocks according to the feedback and verification needed.

AGILE LEADERS
Training Center

How can TDD be introduced into legacy code?

TDD enters legacy code through characterization tests, identified test seams, and small behavior-preserving changes. Teams first capture current behavior, isolate dependencies, and then add new behavior through short test-driven cycles.

Conclusion

Participants take back a TDD Practice Portfolio. It combines test lists, unit tests, failure diagnostics, test-double choices, characterization tests, refactoring sequences, pipeline feedback, and a team working agreement. The portfolio helps teams turn software behavior into executable design feedback and safer code changes.