Agile Programming Techniques

3 DAYS

Intended Audience

This expert level course is for:

- Developers who wish to design and develop systems using Agile techniques.

Prerequisites

As the practical work will be done in Java or C#, participants need a working knowledge of one of those languages.

This course covers Agile techniques in practice and participants need to have an understanding of the fundamentals of Agile. We recommend attending Agile Fundamentals course to gain the appropriate background knowledge.

Certification

This course covers the first step in the Agile Development Track of the IC Agile pathway and covers all the learning objectives of the Agile Programming Certification (ICP-PRG). The ICP-PRG Certification is granted on the successful completion of this course.

Overview

During this three day ICAgile accredited course, participants will encounter, and practice, the tools and techniques for designing and implementing systems. Specifications written in the form of user examples and unit tests will be used to drive development with TDD (Test-Driven Development), BDD (Behaviour-Driven Development) and ATDD (Acceptance Test-Driven Development).

Learning Outcomes

- Create unit tests to test individual classes and modules in isolation
- Safely refactor legacy code bases without breaking existing code
- Drive design and development with unit tests using TDD and BDD
- Write automated specifications/acceptance tests.

Content

- **Unit Testing and Dependency Injection**

Learn about the anatomy of a unit test and create tests in JUnit or NUnit. Use common patterns for creating and structuring tests and fixtures to: create readable and maintainable tests; and avoid duplication. Create test doubles using stubs and mocks to isolate the test fixture and eliminate any dependencies. Apply dependency injection for easy configuration to facilitate the use of test doubles without needing test hooks in production code.

- **Refactoring**

Apply refactoring patterns to safely restructure code into a better design without changing its outward behaviour. Discover code smells identifying code that should be refactored in order to remove technical debt. Look for seams to add unit tests to legacy code in a safe fashion without refactoring the code, even though the code was not designed with testing in mind.

- **Design**

Understand the need for simplicity in design and code. Learn to evaluate design and design principles from the perspective of simplicity and quality. Look at how systems decay over time and discover techniques to monitor the technical debt. Use the principle of simplicity of design and code to avoid or eliminate technical debt.

- **TDD and BDD**

Discover how test-driven development focusses on the design and the desired result. Apply TDD to create a high-quality system. Understand the difficulties of applying TDD. Use behavioural requirements to identify what to do next and what to test.

- **The Build Process**

Understand how Agile collaboration with concepts such as collective accountability and collective ownership. Experience pair programming, identifying the benefits and challenges and understanding the different types of pairing. Look at how build tools provide automation of the build and test process, and how version control provides a source code repository to support multiple developers working on the same system.
Learn how continuous integration helps us have confidence regarding the quality of the code in the source code repository and manages integrating the work of multiple developers.

- **ATDD**

Discover how TDD delivers technical quality but not necessarily user quality. Apply ATDD to ensure user quality and drive TDD. Create readable user specifications that can be "executed" as automated acceptance tests, resulting in "living documentation".