TDD with JUnit

Course Summary

Description

Our TDD course provides an introduction to one of the most successful Agile practices to date, taught using either JUnit or TestNG.

Not only will you learn about unit testing in general, but specifically about a development practice that puts testing first and almost always yields significant improvements in developer productivity, quality, and satisfaction. Developers will be encouraged and hopefully inspired to write good, testable code in a fraction of the time required by more "common" techniques.

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

- Understand how Test Driven Development fits into the development lifecycle
- Understand how a test-first appoach can improve their productivity
- Develop production code faster
- Develop production code with significantly less defects
- Develop software that can be tested
- Recognize and apply best practices
- Use JUnit more effectively
- Use supplementary testing frameworks for mocking
- Test hard-to-test code
- Recognize anti-patterns in production code that make testing difficult
- Refactor code with more confidence

Topics

- Test Driven Development
- Testing, Design, and Testing Patterns
- Testing hard to test code
- Optional

Audience

This course is designed primarily for Java developers. It is suitable for quality-assurance engineers, and development managers.

Prerequisites

- Solid understanding of Java technology is needed to get the full benefit of this course
- Prior familiarity with testing concepts (e.g. unit, integration, functional, load) is recommended

Duration

Three days

TDD with JUnit

Course Outline

I. Test Driven Development
- A. Why developers tend to not write tests
- B. Why developers love writing tests
- C. Why we test (it's not just about finding bugs)
- D. The Test Driven Development Lifecycle
- E. Exercise: Reducing code-complete time by writing tests first
- F. Striking a balance between testing and coding and finding synergy
- G. Best Practices for test driven development

II. Testing, Design, and Testing Patterns
- A. Design for testability
- B. Validating design through testing
- C. Rule of Three
- D. Dependency Injection
- E. Mock Objects
- F. Test Stubs
- G. Test Fixtures

III. Testing hard to test code
- A. Technical Debt
- B. Refactoring
- C. Testability Antipatterns:
- D. Static Variables and Mutable Global
- E. State Large Functions
- F. Highly Coupled Modules
- G. PowerMock

IV. Optional
- A. Other Testing Frameworks (TestNG)
- B. Other Mocking Frameworks (Mockito)
- C. Performance Testing
- D. GUI Testing (FEST)
- E. Web GUI Testing (Selenium)
- F. Testing with the Spring Framework