Java Test Driven Development with TestNG

Course Summary

Description

Test Driven Development (TDD) has become a standard best practice for developers, especially those working in an Agile development environment. TDD is more than just automated unit testing, it is a team and individual development discipline that, when followed correctly, increases productivity of both individual developers and entire teams. From the programmer's perspective, TDD has another benefit – it allows programmers to eliminate the tedious tasks of debugging and reworking code so that programmers can focus on the creative work of designing and writing code. It makes programming fun again.

The course integrates two primary learning streams.

The first is the how to effectively implement TDD in a production or development environment and integrating TDD practices with other practices like software craftsmanship, agile design practices, continuous integration and best practices in object oriented programming and Java development.

The second learning stream is an in depth and hands on deep dive into Java TDD tools such as mocking libraries, matching libraries and the TestNG framework itself. TestNG (Test Next Generation) is a test framework inspired by JUnit and NUnit but with additional features and functionality
The class is designed to be about 50% hands on labs and exercises, about 25% theory and 25% instructor led hands on learning where students code along with the instructor.

Topics

- The TDD process - "red, green, refactor"
- Eliminating technical debt with TDD
- Why TDD works
- Integrating the TDD discipline into a development process
- Using TDD to support programming and design best practices
- Developing a TDD project using TestNG
- TestNG concepts, architecture and features
- Organizing and managing tests using testng.xml
- Assertion libraries (hamcrest, etc.)
- Using Mocks effectively
- Mocking libraries (Mockito, JMockit, EasyMock, etc.)
- How to developing good tests and test suites
- Best practices when using TDD and JUnit to improve development
- Code smells and refactoring
- Using TDD to refactor code
- Migrating to TDD as a programming discipline

Audience

This course is designed for Java programmers.

Prerequisites

Before taking this course, students should have a good knowledge of Java and have at least an intermediate programming skill level.

Duration

Three days

ProTech protechtraining.com *"Charting the Course ... ... to Your Success!"*

Java Test Driven Development with TestNG

Course Outline

I. Java Test Driven Development Introduction
- A. The TDD process as a discipline
- B. How TDD improves efficiency and effectiveness of programming
- C. Eliminating technical debt, debugging and rework
- D. Integrating TDD with best practices in program design and coding
- E. TDD as a a core Agile practice
- F. The Agile testing quadrants
- G. The importance of test automation
- H. Refactoring: what it is and why we do it

II. An Introduction to TestNG
- A. TestNG architecture and functionality
- B. The TDD process implemented with TestNG
- C. Runners, fixtures and test execution
- D. Using the testing.xml file to run tests
- E. TestNG assertions – how tests pass and fail
- F. TestNG annotations
- G. Writing and validating a TestNG test
- H. Introduction to matchers and mocks

III. TDD Best Practices I
- A. Testing through interfaces
- B. Command / Query segregation
- C. Functional testing concepts
- D. Relationship between good class design and ease of testing
- E. Understanding Unit testing – component isolation
- F. Planning the development to minimize work
- G. How to decide what test to add next
- H. Errors, faults, failures and exceptions
- I. TDD best practices
- J. Common errors when implementing TDD

IV. More JUnit
- A. Design by contract
- B. Writing tests for preconditions, post-conditions and invariants
- C. Testing exceptions with TestNG
- D. Using testNH test fixtures effectively
- E. Selective execution of tests and grouping of tests
- F. Ordering the addition of of tests into TestNG
- G. Differences between test errors and test failures

V. Testing Concepts
- A. Criteria for good testing: validity, accuracy and reliability
- B. Why our tests have to be correct
- C. Common sources of test case errors
- D. Systematic and algorithmic test case development
- E. Deriving test cases from acceptance tests
- F. Functional coverage measures
- G. Determining optimal numbers of tests
- H. Using test cases to identify requirements and specification problems
- I. Dealing with valid, invalid and outlier test cases
- J. Combinatorial versus stateful testing

VI. Assertions and Predicates
- A. Four generations of assertions: Java, JUnit, Hamcrest and AssertJ
- B. Using hamcrest to develop complex assertion predicates
- C. Using hamcrest to examine structures, lists, etc
- D. Writing complex predicates in AssertJ
- E. Overview of hamcrest and AssertJ features

Java Test Driven Development with TestNG

Course Outline (cont'd)

VII. Mocking and Mock Libraries

A. Mocks, stubs, drivers and component unit testing
B. Implementing interfaces with a mock object
C. Overview of mocking libraries: EasyMock, Mockido, JMockit ec.
D. How mocking libraries work
E. Implementing and using mocks from a mocking library
F. Designing and implementing a mocking strategy

VIII. TDD Best Practices II

A. Organizing and maintaining the test environment
B. Testing the test code
C. Developing tests and code design together
D. Deciding intensive and comprehensive should be
E. TDD implementation patterns
F. Metrics for TDD

IX. Refactoring

A. Refactoring as controlled code changes
B. Using TDD to implement a refactoring
C. Code smells – driver for refactoring
D. Code refactoring versus design refactoring
E. Refactoring to a design pattern
F. Using refactoring to reduce technical debt
G. Refactoring best practices

X. Implementing TDD

A. Review of TDD best practices
B. Review of JUnit best practices
C. Planning a TDD project
D. Integrating TDD into a development process
E. Integrating TDD with coding excellence
F. The importance of metrics
G. Developing an implementation plan
H. Pitfalls, snares and traps to avoid