Introduction to Test Driven Development and Refactoring with Java

The Test-driven Development course delivers a hands-on view into how TDD can be used by developers, project managers, and the testing teams to create higher quality software.

Ensuring every student has the same understanding of standard software testing procedures, the TDD course begins with an examination of common test terminologies, practices, benefits, and pitfalls. The course then moves into a discussion on the theory and practice of test-driven development, the applicability of TDD in modern software development paradigms and how it can be leveraged within different software engineering processes.

Once the foundation has been laid, the Test-driven Development course transitions into a hands-on learning lab, where students learn the processes and practices of test-driven development in the development of a basic application by following TDD approach.

Objectives

- Understand software testing concepts: Unit test, regression test, integration test, user acceptance tests, continuous testing, etc.
- Understand the Test Driven Development process.
- Follow the test-driven development process with test first approach.
- Understand the TDD cycle Red-Green-Refactor.
- Working with a unit test framework JUnit in eclipse java environment.
- Incorporate TDD into the organization: Advice, tips, tricks, and pitfalls.
- Understand the use of test doubles with Mock frameworks in unit testing.
- Understand the concept of Continuous Integration

Pre-requisites

- The participants must be well versed with core java programming skills.
- Knowledge about the unit testing, integration testing processes.
- Awareness about Eclipse Java development environment.
- Awareness about server side development with JavaEE is an added advantage.

Classroom lab setup

CPU Intel core compatible with minimum 4GB RAM and 180GB HDD with Windows 7/8/10 64 bit system with JDK1.8 64 bit, Eclipse-Jee-2020-03 and Adobe Acrobat Reader to be installed and configured.

The users should have admin privileges on their systems.
Live Internet connection with reasonable speed for updating the patches with download permissions to be made available in class room.

Knowledge Delivery Process

The theoretical topics are discussed interactively and technical details are demonstrated with practical examples. The participants work on the hands on exercises which strengthen the concepts learned.

Each topic is supplemented with practical demonstrations and exercises for the participants.

Course Duration: 2 days.

Course Instructor: Prakash Badhe.

Course Outline
This day wise course plan is based on the inputs from the client team.

Day1
~~Review: Traditional software development process~~

- ~~Waterfall model~~
- ~~Tests to satisfy the code!~~
- ~~Pros and cons~~
- ~~Limitations and Challenges~~
- ~~Manage the change~~
- ~~Customer view of the application~~

~~How to Smoothen the code development~~

- ~~Iterative development with smaller increments~~
- ~~Robust testing of code~~
- ~~Refactor the code and regression testing~~
- ~~Automate the build process~~
- ~~Focus on smaller increments~~
- ~~Get the quick feedback and correct it~~
- ~~Continuous Testing~~
- ~~Get the maximum code coverage~~

Unit testing process

- The unit under test
- Actions on the unit
- Verify the behaviour of unit under test
- Check for failure or success in the test.
- Go to the next test and continue.

Unit Testing for Java

- The xUnit for unit test specifications
- Unit testing tools and libraries overview
    - JUnit
    - JUnit libraries as JUnitPerf, junit-dataprovider, CallbackParams, ContiPerf etc.
    - TetsNG
    - Spock
    - Cactus for server side testing
    - Mock libraries

~~Java GUI Testing tools overview~~

- ~~Abbot for Java UI testing~~
- ~~Jemmy for UI actions~~
- ~~SWTBot for Eclipse UI Testing~~

JUnit: Unit Test Framework for Java

- xUnit test implementation
- JUnit architecture and capabilities
- JUnit Test Annotations
- The JUnit test life cycle.
- TestCase and Test-Suites
- Overview on integration with other tools

Test-Driven Development Process

- Test Driven development – Code to satisfy the test cases.
- Why Adopt TDD
- Where to start with TDD
- The TDD process.
- The tests **drive** the code.
- The TDD cycle : Red-Green-Refactor
- Implement a case study in TDD manner.

Start application development in TDD way

- Case study introduction and overview
- Understand the requirements
- Start with lowest minimum component.
- Write test case and let it fail first time.
- Write minimum code to make the test success.
- Write more test cases and write minimum code to satisfy the test cases.
- Regression testing
- Evolve the application by TDD process.
- Refactoring the code to make it maintainable
- Best practices for unit testing.

Test code with Mock objects

- Isolate the dependencies during test
- Abstraction and isolation
- Use of Fakes and Test Doubles for isolated testing
- Testing in isolation with mock objects
- The java mock libraries overview
- The Mockito architecture
- Use mock objects with Mockito
- Record-Play-Verify
- Mocks as stubs and proxies
- Assert and Verify test Invocations
- Test the objects in absence of dependencies with mock objects.

- Mock the database objects and services

Data Driven testing

- Parameterized test cases
- Re-use test cases
- Use of JUnitParam
- Load test data from Excel/CSV files

Day2

Test Multi-Threaded code

- Challenges
- Concurrent-junit library usage
- Testing with Asynchronous code
- Use of Call backs
- Test with ConcurrentUnit

Code Refactoring Process

- Refactoring Concepts and goals
- Change the code structure without affecting functional behaviour
- When to refactor
- Avoid code duplications
- Make the code loosely coupled with code abstractions
- Make it more reusable and flexible.
- Make it maintainable
- How to refactor the code
- Re-test the re-factored Code
- Apply Design Patterns in refactoring process
- SOLID Principles overview

Overview: Refactoring techniques

- Make the code more flexible and readable.
    - Extract variable
    - Extract method
    - Extract class
    - Extract interface
    - Extract super class
    - Re-naming of variables, classes and methods.
    - Shorten the long method name
    - Reduce the method parameters
    - Global to local or vice versa for variables scope
    - Push-pull of methods and variables to base
    - Add abstract base class
- Overview on working with legacy code

The Test-Driven Development Impact

- Customer view of application in stages
- Customer involvement
- Change management.
- Developer confidence
- Easier Fault isolation
- Scope of automation and regression testing
- Better code coverage
- Scope for quality coding and performance
- Dos and Don’ts for TDD

Spock Test Framework Introduction

- BDD grammer of test specifications
- Spock features
- Groovy usage
- Spock Web Console Usage
- Setup and Declarative configuration
- Define test specifications with blocks
- Assert with groovy
- Assert exceptions
- Data Driven testing
- Use of Data table
- Mocking with Spock

Question-Answers session

****