Business Skills - Test-Driven Development Workshop

Code: TDDWS
Length: 3 days
URL: View Online

Test-Driven Development (TDD) is a design engineering process that relies on a very short development cycle. A TDD approach to software development requires a thorough review of the requirements or design before any functional code is written. The development process is started by writing the test case, then the code to pass the test and then refactoring until completion. Benefits of a TDD approach to software engineering include faster feedback, higher acceptance, reduced scope creep and over engineering, customer centric and iterative processes, and modular, flexible, maintainable code. This 3-day instructor led course is a deep dive in to TDD that incorporates the steps that are necessary for effective implementation. Participants will cover Unit Tests, User Stories, Design, Refactoring, Frameworks, and how to apply them to existing solutions. In addition, this course explores the implications of code dependencies, fluid requirements, and early detection of issues. This is an interactive class with hands-on labs. To get the most out of this course, students are encouraged to fully participate. This course demonstrates the skills developers and teams need for building quality applications sustainably, with quality, for the life of the code base.

Skills Gained

- Provide knowledge and understanding of Unit Testing principles and practices
- Understand the role of Unit Testing in software development and testing
- Write effective Unit Testing
- Properties of effective unit tests
- How to use mock objects to isolate the “system under test”
- Effective refactoring of the code base
- Benefits of the test-first and Test-Driven Development
- Techniques and practices to aid in the successful adoption of Test-Driven Development
- How to use Acceptance Testing and Behavior-Driven Development to further advance Test-Driven Development

Who Can Benefit

- Software Developers and Programmers
- Agile Practitioners
- Quality Assurance Professionals
- Software Testers
- Product Owners
- Project Managers
- IT Managers
- Software Engineers

Course Details

Module 1: Agile Overview

Test Driven Development is a key component of the Agile Software Development Methodology and of the overall DevOps movement. So it is helpful to have at a minimum a high level understanding of Agile practices and scrum ceremonies and TDD fits into the overall Agile, Scrum and DevOps landscape. Module 1 serves as a leveling exercise to ensure that team member are speaking the same language during upcoming labs and discussions.

- Lab: Explore the Board of an Agile Project

What is Agile Software Development

- DevOps Overview
- The Agile Manifesto
- Scrum vs Agile

Components of Agile

- User Stories
- Tasks
- Bugs
- Automated Builds
- Automated Tests
- Continuous Inspection

The Role of TDD in Agile Development

- Automated Unit Tests
- Automated Acceptance Tests

Benefits of Agile Development

- Value Chain
- Change Management
- Feedback

Becoming Agile

- Common Implementation Issues
- Agile/DevOps Culture
- Continuous Improvement
- Kanban Board
- User Stories
- Tasks
- Bugs
- Work in Progress
- Burndown Chart

Module 2: Principles of Agile Development

TDD is directly influenced by design, so it will be a priority to take this into context during implementation. Considering design principles will enable teams to experiment with different approaches, and gear towards more functional programming.

- Lab: Interfaces and Abstract Classes

Design Principles Overview

- Encapsulation
- Interface not implementation
- Composition not Inheritance
- Open Closed Principle
- Principle of Least Knowledge
- Extend not Edit

Coding Principles

- KISS
- YAGNI
- OOO
- DRY
- PIE
- Explore an abstract class
- Explore a basic Interference
- View implemented interface on a derived class

Module 3. Unit Testing

Unit Testing is a critical component of Test Driven Development (TDD). Small units of code are tested throughout the development process, which isolates functionality to ensure that individual parts work correctly.

- Lab: Unit Test Attributes and Assertions

Unit Test Fundamentals

- Reason to do Unit Testing
- What to Test: Right BICEP
- CORRECT Boundary Conditions
- Properties of Good Tests

Advanced Unit Testing

- Unit Testing Legacy Applications
- Techniques for Legacy Code
- External Dependencies
- Unit Testing the Database

Frameworks

- What is NUnit
- NUnit Building Blocks
- Test Cases
- Test Suite
- Test Fixture
- Examples

Test Runners

- What is Test Runner
- Assertions
- "Fluent" Assertions
- Examples

Advanced Test Attributes

- Setup / TearDown
- SetupFixture / TearDownFixture
- NUnit Lifecycle
- ExpectedException
- Ignore
- Explore Sample Unit Tests
- Test Attributes
- Assert Statements

Module 4: Test Driven Development

Essential TDD techniques require developers write programs in short development cycles, and there are critical steps that must be taken. Tests are created before the code is written. Once the code passes testing, it is refactored to adhere to the most effective and acceptable standards.

- Lab: Test Driven Development

TDD Rhythm

- TDD Overview
- Red, Green, Refactor
- TDD Benefits

Sustainable TDD

- Development without TD
- Test Last
- Test Last in Parallel
- Test Driven Development

Supporting Practices

- Collective Ownership

- Continuous Integration
- Pair Programming

Pair Programming

- Pairing Setup
- Resistance
- Results: Time
- Results: Defects
- How it works
- Advantages of Pairing
- Challenges
- Pair Rotation

Pairing Techniques

- Ping Pong Pairing
- Promiscuous Pairs
- Pair Stairs
- Cross-functional Pairing

Eight Wastes of Software Development

- Ripple effect of defects
- Partially Done Work
- Extra Features
- Relearning
- Handoffs
- Task Switching
- Delays
- Defects

Test Automation

- Automate, Automate, Automate
- Automate Early and Often
- Additional Topics Identified
- Create a test project
- Create stack tests
- Create minimal project structure
- Write code to pass tests

Module 5: Refactoring

Refactoring is another essential technique of TDD, and most software development teams are most likely doing some form of refactoring regularly. Refactoring can be used in a number of different workflows which will be explored in this module.

- Lab: Refactoring for Optimization and Maintenance

Why Refactor?

- Red, Green, Refactor
- Benefits
- Development without TDD

Refactoring Methods

- Test Last Approach
- Test Last in Parallel
- Test-Driven Design
- Sustainable TDD

Refactoring Cycle

- Reduce Local Variable Scope
- Replace Temp with Query
- Remove Dead Code
- Extract Method
- Remove Unnecessary Code
- Collapse Loop
- Extract method

Module 6: Pair Programming

Pair Programming is an effective way to improve code quality. In this module, we will discuss pairing and how it leads to better software design, and a lower cost of development.

Pair Programming

- Pairing Setup
- Results: Time
- Results: Defects
- How it works

Advantages of Pairing

- Both Halves of the Brain
- Focus
- Reduce Interruptions
- Reduce Resource Constraints
- Multiple Monitors
- Challenges

Pairing Techniques

- Pair Rotation
- Ping Pong Pairing

- Promiscuous Pairs
- Pair Stairs
- Cross-Functional Pairing

Module 7: Acceptance Testing & BDD

Acceptance Tests are an important form of functional specification, and Behavior Driven Development dictates what happens as an effect of these tests being run. In this Module, we will cover the difference between them, and how they are closely related.

- Lab: Acceptance Testing
- Lab: Automate Acceptance Tests

Acceptance Testing

- Acceptance Tests
- Why Acceptance Tests?
- Acceptance Test Execution
- Who Writes Acceptance Tests
- Pair Test Writing
- Acceptance Test Writing Exercise

Best Practices for Effective Testing

- Keys to Good Acceptance Tests
- Writing Acceptance Criteria
- Acceptance Test Example
- Acceptance Test-Driven Development (ATDD)

BDD vs. ATDD

- Specification by Example
- BDD Frameworks
- BDD Examples
- Queue Acceptance Test
- Automate Execution and Bug Reporting

Module 8: Principles & Benefits

In this module, we will take a look at the benefits of TDD, the principles that drive them. We will review the collaborative nature of TDD, and focus on the communicative process that makes it work.

Consequences of no Testing

- Ripple Effect of Defects
- Partially Done Work
- Extra Features
- Relearning
- Handoffs
- Task Switching

- Delays

TDD Solutions

- Continuous Unit Testing
- Collective Ownership

Module 9: Unit Test Examples

In order to implement Unit Tests efficiently, it is critical that developers understand their properties.

- Lab: Queues and Stacks

Queues and Stacks

- Implementing a Queue
- Enqueue and Dequeue Methods
- Peek Method
- Create a Test List

Unit Test Examples

- More Tests?
- The Sieve of Eratosthenes
- Algorithm
- Understanding the Sieve

Advanced Refactoring

- Replace Nested Conditional with Guard Clauses
- Inline Method
- Rename Variable
- Collapse Loops
- Remove Dead Code
- Implementing queues tests
- Implementing queues
- Implementing stack tests
- Implementing stack class

Module 10: Testing for Non-Developers

- Lab: Fit and Fitnesse

FIT and Fitnesse

- FI
- Fitnesse
- FIT & Fitnesse
- Core FIT Fixtures
- Using a Column Fixture

- Fitnesse Wiki Syntax
- Create a Column Fixture

Additional TDD Considerations

- Using a RowFixture
- Using a ActionFixture
- SLIM
- SLIM Tables
- The Gherkin Language
- Step Definitions
- Install fit and fitnesse
- Create rowfixture test
- Create actions fixture test
