Skillsoft
global knowledge™

TEST-DRIVEN DEVELOPMENT WORKSHOP WITH JAVA

Course Code: 4281

Learn to implement Test-Driven Development methods by incorporating unit testing, design, refactoring and frameworks into your workflow and how to apply them to existing solutions.

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 TDD training course instructor is a deep dive in to Test Driven Development 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.

What You'll Learn

In this TDD Training Course, You Will Learn:

- The role of Unit Testing in software development and testing
- How to write effective Unit Testing
- The properties of effective unit tests
- How to use mock objects to isolate the “system under test”
- Effective refactoring of the code base
- The 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 Needs to Attend

This class is designed for professionals who have experience in language programming, including C+ or Java. Specific titles of attendees include:

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

Skillsoft global knowledge™

TEST-DRIVEN DEVELOPMENT WORKSHOP WITH JAVA

Course Code: 4281

CLASSROOM LIVE	$3,250 CAD	3 Day

Classroom Live Outline

Part 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. Part 1 serves as a leveling exercise to ensure that team member is speaking the same language during upcoming labs and discussions.

1. What is Agile Software Development
    - DevOps Overview
    - The Agile Manifesto
    - Scrum vs Agile
2. Components of Agile
    - User Stories
    - Tasks
    - Bugs
    - Automated Builds
    - Automated Tests
    - Continuous Inspection
3. The Role of TDD in Agile Development
    - Automated Unit Tests
    - Automated Acceptance Tests

Lab: Explore the Board of an Agile Project

- Kanban Board
- User Stories
- Tasks
- Bugs
- Work in Progress
- Burndown Chart

Part 2: 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.

1. Unit Test Fundamentals
    - Reason to do Unit Testing
    - What to Test: Right BICEP
    - CORRECT Boundary Conditions
    - Properties of Good Tests
2. Frameworks
    - What is JUnit
    - JUnit Building Blocks
    - Test Cases
    - Test Suites
    - Examples
3. Agile Testing Strategy
    - Agile Testing Quadrant
    - Automation Pyramid
    - Assertions
4. Test Attributes
    - Setup / TearDown
    - JUnit Lifecycle
    - System Under Test
    - Test Design Strategy
    - Naming our Tests
    - Exceptions

Lab: Introduction to Unit Testing

- IDE and Project Setup
- Running our first Unit Test
- Explore Junit framework
- Test Attributes
- Assert Statements

Part 3: 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.

1. TDD Rhythm
    - TDD Overview
    - Red, Green, Refactor
    - TDD Benefits
2. Sustainable TDD
    - Development without TD
    - Test Last

    - Test Last in Parallel
    - Test Driven Development
3. Supporting Practices
    - Collective Ownership
    - Continuous Integration
4. Eight Wastes of Software Development
    - Ripple effect of defects
    - Partially Done Work
    - Extra Features
    - Relearning
    - Handoffs
    - Task Switching
    - Delays
    - Defects
5. Test Automation
    - Automate, Automate, Automate
    - Automate Early and Often
    - Additional Topics Identified

Lab: Test Driven Development

- Start Test Driven Development on our example App
- Write unit test cases
- Experience RED, Green, Refactor Process

Part 4: 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.

1. Design Principles Overview
2. Coding Principles
3. isolation of the SUT
4. Developing independently testable units
5. Test doubles
    - Introducing test doubles
    - Stubs
    - Fakes
    - Mocks

Lab: Continue development on example App

- Setting up Test doubles for our example app
- Discuss and implement test dummy and test stubs

Part 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 Part.

1. Why Refactor?
    - Red, Green, Refactor
    - Benefits
    - Development without TDD
2. Refactoring Methods
3. Refactoring Cycle
    - Reduce Local Variable Scope
    - Replace Temp with Query
    - Remove Dead Code
    - Extract Method
    - Remove Unnecessary Code

Lab: Continue our example project

- Implement new test cases
- TDD Cycle
- Discuss and implement Refactoring Needs

Part 6: Pair Programming

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

1. Pair Programming
    - Pairing Setup
    - Results: Time
    - Results: Defects
    - How it works
2. Advantages of Pairing
    - Both Halves of the Brain
    - Focus
    - Reduce Interruptions
    - Reduce Resource Constraints
    - Multiple Monitors
    - Challenges
3. Pairing Techniques
    - Pair Rotatio
    - Ping Pong Pairing
    - Promiscuous Pairs
    - Pair Stairs
    - Cross-Functional Pairing

Lab: Experience pair programming and continue developing our example app

Part 7: Acceptance Test Driven Development (ATDD) & Behavior-Driven Development (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 Part, we will cover the difference between them, and how they are closely

related.

1. Acceptance Testing
    - Acceptance Tests
    - Why Acceptance Tests?
    - Acceptance Test Execution
    - Who Writes Acceptance Tests
    - Pair Test Writing
2. Best Practices for Effective Testing
    - Keys to Good Acceptance Tests
    - Writing Acceptance Criteria
    - Acceptance Test Example
    - Acceptance Test-Driven Development (ATDD)
3. BDD vs. ATDD
    - Specification by Example
    - BDD Frameworks
    - BDD Example**s**

Lab: Experience ATDD and BDD
• Experience ATDD or BDD and discuss the impact to TDD

Part 8: Advanced TDD
In order to implement Unit Tests efficiently, it is critical that developers understand their properties.
Lab: Demonstration for Mocking/doubles with our example App

1. TDD Solutions
    - Continuous Unit Testing
    - Collective Ownership
2. Advanced Unit Testing
    - Unit Testing Legacy Applications
    - Techniques for Legacy Code
    - External Dependencies
    - Mocking frameworks
    - Unit Testing the Database
3. Outside In vs Inside Out
    - Working with database
    - Working with mocking frameworks
4. Test Strategy
    - Continues Integration
    - Batch Execution of TestCases
5. Unit Test Examples
    - More Tests
    - Algorithm
6. Advanced Refactoring

Lab: Demonstration for Mocking/doubles with our example App

- Working with database

- Working with mocking frameworks

Part 9: Simulation

Experience Agile development with test driven development and pair programming

Skillsoft global knowledge™

TEST-DRIVEN DEVELOPMENT WORKSHOP WITH JAVA

Course Code: 4281

VIRTUAL CLASSROOM LIVE	$3,250 CAD	3 Day

Virtual Classroom Live Outline

Part 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. Part 1 serves as a leveling exercise to ensure that team member is speaking the same language during upcoming labs and discussions.

1. What is Agile Software Development
    - DevOps Overview
    - The Agile Manifesto
    - Scrum vs Agile
2. Components of Agile
    - User Stories
    - Tasks
    - Bugs
    - Automated Builds
    - Automated Tests
    - Continuous Inspection
3. The Role of TDD in Agile Development
    - Automated Unit Tests
    - Automated Acceptance Tests

Lab: Explore the Board of an Agile Project

- Kanban Board
- User Stories
- Tasks
- Bugs
- Work in Progress
- Burndown Chart

Part 2: 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.

1. Unit Test Fundamentals
    - Reason to do Unit Testing
    - What to Test: Right BICEP
    - CORRECT Boundary Conditions
    - Properties of Good Tests
2. Frameworks
    - What is JUnit
    - JUnit Building Blocks
    - Test Cases
    - Test Suites
    - Examples
3. Agile Testing Strategy
    - Agile Testing Quadrant
    - Automation Pyramid
    - Assertions
4. Test Attributes
    - Setup / TearDown
    - JUnit Lifecycle
    - System Under Test
    - Test Design Strategy
    - Naming our Tests
    - Exceptions

Lab: Introduction to Unit Testing

- IDE and Project Setup
- Running our first Unit Test
- Explore Junit framework
- Test Attributes
- Assert Statements

Part 3: 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.

1. TDD Rhythm
    - TDD Overview
    - Red, Green, Refactor
    - TDD Benefits
2. Sustainable TDD
    - Development without TD
    - Test Last

    - Test Last in Parallel
    - Test Driven Development
3. Supporting Practices
    - Collective Ownership
    - Continuous Integration
4. Eight Wastes of Software Development
    - Ripple effect of defects
    - Partially Done Work
    - Extra Features
    - Relearning
    - Handoffs
    - Task Switching
    - Delays
    - Defects
5. Test Automation
    - Automate, Automate, Automate
    - Automate Early and Often
    - Additional Topics Identified

Lab: Test Driven Development

- Start Test Driven Development on our example App
- Write unit test cases
- Experience RED, Green, Refactor Process

Part 4: 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.

1. Design Principles Overview
2. Coding Principles
3. isolation of the SUT
4. Developing independently testable units
5. Test doubles
    - Introducing test doubles
    - Stubs
    - Fakes
    - Mocks

Lab: Continue development on example App

- Setting up Test doubles for our example app
- Discuss and implement test dummy and test stubs

Part 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 Part.

1. Why Refactor?
    - Red, Green, Refactor
    - Benefits
    - Development without TDD
2. Refactoring Methods
3. Refactoring Cycle
    - Reduce Local Variable Scope
    - Replace Temp with Query
    - Remove Dead Code
    - Extract Method
    - Remove Unnecessary Code

Lab: Continue our example project

- Implement new test cases
- TDD Cycle
- Discuss and implement Refactoring Needs

Part 6: Pair Programming

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

1. Pair Programming
    - Pairing Setup
    - Results: Time
    - Results: Defects
    - How it works
2. Advantages of Pairing
    - Both Halves of the Brain
    - Focus
    - Reduce Interruptions
    - Reduce Resource Constraints
    - Multiple Monitors
    - Challenges
3. Pairing Techniques
    - Pair Rotatio
    - Ping Pong Pairing
    - Promiscuous Pairs
    - Pair Stairs
    - Cross-Functional Pairing

Lab: Experience pair programming and continue developing our example app

Part 7: Acceptance Test Driven Development (ATDD) & Behavior-Driven Development (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 Part, we will cover the difference between them, and how they are closely

related.

1. Acceptance Testing
    - Acceptance Tests
    - Why Acceptance Tests?
    - Acceptance Test Execution
    - Who Writes Acceptance Tests
    - Pair Test Writing
2. Best Practices for Effective Testing
    - Keys to Good Acceptance Tests
    - Writing Acceptance Criteria
    - Acceptance Test Example
    - Acceptance Test-Driven Development (ATDD)
3. BDD vs. ATDD
    - Specification by Example
    - BDD Frameworks
    - BDD Example**s**

Lab: Experience ATDD and BDD
• Experience ATDD or BDD and discuss the impact to TDD

Part 8: Advanced TDD
In order to implement Unit Tests efficiently, it is critical that developers understand their properties.
Lab: Demonstration for Mocking/doubles with our example App

1. TDD Solutions
    - Continuous Unit Testing
    - Collective Ownership
2. Advanced Unit Testing
    - Unit Testing Legacy Applications
    - Techniques for Legacy Code
    - External Dependencies
    - Mocking frameworks
    - Unit Testing the Database
3. Outside In vs Inside Out
    - Working with database
    - Working with mocking frameworks
4. Test Strategy
    - Continues Integration
    - Batch Execution of TestCases
5. Unit Test Examples
    - More Tests
    - Algorithm
6. Advanced Refactoring

Lab: Demonstration for Mocking/doubles with our example App

- Working with database

- Working with mocking frameworks

Part 9: Simulation

Experience Agile development with test driven development and pair programming

TEST-DRIVEN DEVELOPMENT WORKSHOP WITH JAVA

Course Code: 4281

PRIVATE GROUP TRAINING	3 Day

Visit us at www.globalknowledge.com or call us at 1-866-716-6688.

Date created: 4/2/2026 4:38:42 AM