Course Outline

ProTech Professional Technical Services, Inc.

Test Driven Development for COBOL Programmers

Course Summary

Description

This course is a three-day introduction to developing COBOL code using the Test-Driven Development (TDD) methodology. TDD is both a team and individual code development process which measurably increases the productivity of individual developers and entire teams. TDD allows programmers to eliminate the tedious tasks of debugging and reworking code so that they can focus on the creative work of designing and writing code. Many programmers in a wide range of programming languages have embraced TDD because" it makes programming fun again."

TDD is now starting to make in roads in the COBOL programming community as more organizations start to adopt different Agile and DevOps process, including TDD. This course is designed to help COBOL developers make the transition into using TDD in a way that is specifically tailored for COBOL development and maintenance programming.

The course focuses on how to set up and use the TDD process for both new application development and for supporting modifications of existing COBOL code; including how to integrate TDD into application development or maintenance activities. The best practices for TDD are covered in depth supported by practical examples. A main topic in the course is an exploration of how TDD is used to refactor and improve existing COBOL code.

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. The course closes with students developing an action plan for implementing what they have learned in class into their own development environment.

The hands-on portion of the course in OS and vendor agnostic. GnuCOBOL is to provide a safe and robust programming environment, but since the content of the course focuses exclusively on the standard COBOL and not on any OS specific implementations, what students learn can be ported into a variety of different main frame and other environments where COBOL is deployed.

Topics

- The TDD process -"red, green, refactor"
- Eliminating technical debt with TDD
- Why TDD works and how it supports high quality code
- Integrating the TDD discipline into a development activities
- TDD and programming and design best practices
- Developing a TDD project using a Unit Testing Framework.
- TDD Tools: concepts, architecture and features
- Assertions and matchers
- Implementing and using mocks effectively
- Best practices for developing good test cases and test suites
- Best practices when using TDD and automated testing to improve development
- Code smells and refactoring
- Improving Code Quality using TDD
- Using TDD to refactor code
- Migrating to TDD as a programming discipline

Audience

This course is designed for intermediate level COBOL programmers.

Prerequisites

Knowledge and experience in COBOL is essential for understanding the course material. No experience or background in software testing is required.

Duration

Three days

Course Outline

Test Driven Development for COBOL Programmers

Course Outline

I. Java Test Driven Development Introduction

A. The TDD process as a code development 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 core Agile practice being migrated to mainframe environments
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 Automated Testing

A. Automated test famework architecture and functionality
B. Test runners, fixtures, test classes and test methods
C. Implementing automated testing for batch processing
D. Assertions – describing how tests pass and fail
E. Writing and validating a unit test case
F. Understanding unit testing versus integration testing
G. Introduction to mock objects and how they are used inunit testing

III. TDD Best Practices I

A. Testing through interfaces and good COBOL design
B. Command / Query segregation
C. Functional testing concepts for TDD
D. Relationship between good program design and ease of testing
E. Understanding Unit testing – component isolation
F. Planning TDD workflow minimize development effort
G. Errors, faults, failures and exceptions
H. Design by contract
I. Writing tests for preconditions, post-conditions and invariants
J. Testing exceptions and for error handling.

IV. The TDD Process

A. Setting up the TDD project
B. Deriving unit tests from acceptance tests
C. Testing the test environment
D. Using test fixtures effectively
E. How to decide what test to add next
F. Red, Green, Refactor
G. TDD best practices
H. Common errors when implementing TDD

V. Testing Concepts

A. Criteria for good testing: validity, accuracy, correctness and reliability
B. Why tests have to be good
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. Mocking, Drivers and Stubs

A. Developing a mock
B. Using a mock
C. Using a mock library
D. Database mocking and developing portable test data
E. Using test data in a mock
F. Testing a mock
G. Modifying and maintaining mocks

VII. 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
F. Using continuous refactoring to reduce technical debt
G. Refactoring best practices
H. Using refactoring for code maintenance

Test Driven Development for COBOL Programmers

Course Outline

VIII. Implementing TDD

A. Review of TDD best practices
B. Unit testing on different COBOL platforms.
C. Planning a pilot TDD project
D. Integrating TDD into a development process (waterfall, Agile or DevOps)
E. Integrating TDD with COBOL code best practices
F. The importance of metrics
G. Developing an implementation plan for migrating to TDD
H. Pitfalls, snares and traps to avoid