Unit Testing in Visual Studio 2019

Course Summary

Description

This course provides students with the knowledge and skills to effectively use Visual Studio 2019 to design, write, and run high-quality .NET unit tests. The course focuses on the applicable features and capabilities of Visual Studio as it relates to unit testing and Test-Driven Development. This course also introduces other popular unit testing tools and techniques, and demonstrates how they integrate with Visual Studio and your team's development lifecycle.

Objectives

By the end of this course, students will be able to:

- Why unit tests are critical to software quality
- How unit tests and integration tests differ
- Popular .NET unit testing frameworks
- Popular JavaScript unit testing frameworks
- MSTest V2 improvements and capabilities
- The anatomy of a unit test
- The 3A pattern (Arrange, Act, Assert)
- Using Assert, StringAssert, and CollectionAssert
- Testing for expected exceptions
- Test class inheritance
- Why and how to test internal APIs
- MSTest, NUnit, and xUnit test projects
- Unit testing .NET Core projects
- Using Test Explorer to manage your tests
- Organizing tests using traits and playlists
- Running unit tests in parallel
- In-Assembly Parallel (IAP) execution
- Parallelism by assembly, class, and method
- Running tests and managing test results
- Viewing, grouping, and filter tests and results
- Creating and using a .runsettings file
- Continuous testing in Visual Studio Test-Driven Development (TDD) as a design practice
- Why write your tests first
- Practicing TDD within Visual Studio
- How to effectively refactor within TDD
- How to effectively refactor legacy code
- Practices for writing good unit tests
- Happy path vs. sad path testing
- Testing boundary conditions (Right-BICEP)
- Organizing tests and test assemblies
- Test naming conventions (e.g. BDD)
- Why and how to analyze code coverage
- Using code coverage as a metric
- Parameterized (data-driven) unit tests
- Concurrent testing using Live Unit Tests
- Concurrent testing using NCrunch (3rd party)
- Testing difficult code with the use of doubles
- Using dummies, fakes, stubs, and mocks
- Using Microsoft Fakes to test difficult code
- Using Rhino Mocks to test difficult code
- Using NSubstitute to test difficult code
- Generating MSTest unit tests with IntelliTest
- Generating NUnit unit tests with IntelliTest

Unit Testing in Visual Studio 2019

Course Summary (cont.)

Topics

- Unit Testing in .NET
- Unit Testing in Visual Studio
- Test-Driven Development (TDD)
- Writing Good Unit Tests
- Leveraging Visual Studio
- Testing Difficult Code

Audience

This course is intended for current software development professionals who are involved with building high- quality .NET applications. Students will use Visual Studio while learning how to design, write, and run unit tests. They will also learn many relevant practices and techniques, such as TDD, refactoring, and how to test difficult code using doubles

Prerequisite

Before attending this course, a student should have experience or familiarity with:

- The C# language
- Visual Studio 2015, 2017, or 2019
- Writing, debugging, and maintaining code
- Application Lifecycle Management basics
- Their organization's development lifecycle
- Building a high-quality software product

Duration

Two Days

Unit Testing in Visual Studio 2019

Course Outline

I. *Unit Testing in .NET*

This module introduces the concepts of unit testing and how it is supported by various .NET unit testing frameworks, including MSTest V2 and NUnit.

A. What is (and isn’t) a unit test
B. Why write unit tests
C. .NET unit testing frameworks
D. MSTest V2, NUnit, xUnit
E. The anatomy of a unit test
F. Writing and running your first unit test

II. *Unit Testing in Visual Studio*

This module introduces Visual Studio test projects, Test Explorer and other testing windows, and the practices for effectively writing and running unit tests and managing test results.

A. Testing support in Visual Studio
B. MSTest, NUnit, and xUnit test projects
C. Test Explorer and other windows
D. Writing and running unit tests in Visual Studio
E. Managing a large number of tests and test results
F. Organizing tests by grouping, filtering, and playlists
G. Continuous testing in Visual Studio

III. *Test-Driven Development (TDD)*

This module introduces Test Driven Development (TDD) and the business case for why you should practice it. Refactoring as well as a discussion of how to work with legacy code are also part of this module.

A. TDD overview and benefits
B. Practicing TDD within Visual Studio
C. Effectively refactoring code
D. Working with legacy code
E. Using CodeLens to support TDD and refactoring

IV. *Writing Good Unit Tests*

Just knowing how to write unit tests and being disciplined in TDD is not enough. This module introduces other practices for ensuring that you write high-quality unit tests that cover more than just the happy path.

A. Asking questions about your code
B. Path testing (e.g. happy, sad, evil, etc.)
C. Right BICEP testing
D. Testing for expected exceptions
E. Maintaining high-quality test code
F. Unit test naming conventions (e.g. BDD)
G. Organizing unit tests

V. *Leveraging Visual Studio*

This module examines additional unit testing features found in Visual Studio, including code coverage, parameterized unit tests, and concurrent testing tools.

A. Analyzing code coverage
B. Using code coverage as a metric
C. Parameterized (data-driven) unit tests
D. DataRow, DynamicData, and DataSource attributes
E. Concurrent testing using Live Unit Testing
F. Concurrent testing using NCrunch

VI. *Testing Difficult Code*

This module introduces tools and techniques for testing difficult code, such as code with runtime dependencies.

A. The need to isolate code under test
B. Doubles (dummies, stubs, fakes, and mocks)
C. Microsoft Fakes framework (stubs and shims)
D. Comparing mocking frameworks
E. Using Rhino Mocks and NSubstitute frameworks
F. Profiling slow running unit tests
G. Using IntelliTest with legacy code