Unit Testing in Visual Studio 2022 (2 Days)

This two-day, instructor-led course provides students with the knowledge and skills to effectively use Visual Studio 2022 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

At course completion, students will have had exposure 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
- 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
- Using Test Explorer to manage your tests
- Organizing tests using traits
- Organizing tests using playlists
- Running unit tests in parallel
- Parallelism by assembly, class, and method
- Running tests and managing test results
- Viewing, grouping, and filtering 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
- Data-driven (parameterized) 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 third-party mocking libraries
- Using moq to test difficult code
- Generating MSTest unit tests with IntelliTest
- Debugging and profiling unit test code

Who Should Attend

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, such as fakes, shims, and mocks.

Prerequisites

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

- The C# language
- Visual Studio 2017, 2019, or 2022
- Writing, debugging, and maintaining code
- Application Lifecycle Management basics
- Their organization's development lifecycle
- Building a high-quality software product

Course Outline

Module 1: Unit Testing in .NET
This module introduces the concepts of unit testing and how it is supported by the various unit testing frameworks.

- What is (and isn't) a unit test
- Why write unit tests
- .NET unit testing frameworks
- MSTest, NUnit, xUnit
- The anatomy of a unit test
- Writing and running your first unit test

Module 2: Unit Testing in Visual Studio
This module introduces Visual Studio test projects, Test Explorer and other testing windows, and the practices for effectively writing, running, and managing unit tests and test results.

- Testing support in Visual Studio
- MSTest, NUnit, and xUnit test projects
- Test Explorer and other windows
- Writing and running unit tests in Visual Studio
- Managing a large number of tests and test results
- Organizing tests by grouping, filtering, and playlists
- Continuous testing in Visual Studio

Module 3: 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.

- TDD overview and benefits
- Practicing TDD within Visual Studio
- Effectively refactoring code
- Working with legacy code
- Using CodeLens to support TDD and refactoring

Module 4: 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.

- Asking questions about your code
- Path testing (e.g. happy, sad, evil, etc.)
- Right BICEP testing
- Testing for expected exceptions
- Maintaining high-quality test code
- Unit test naming conventions (e.g. BDD)

Organizing unit tests Module 5:
Leveraging Visual Studio This module examines additional unit testing features found in Visual Studio, including code coverage, datadriven unit tests, and concurrent testing tools.

- Analyzing code coverage
- Using code coverage as a metric
- Data-driven (parameterized) unit tests
- DataRow, DynamicData, and DataSource attributes
- Concurrent testing using Live Unit Testing
- Concurrent testing using NCrunch

Module 6: Testing Difficult Code This module introduces tools and techniques for testing difficult code, such as code with runtime dependencies.

- The need to isolate code under test
- Doubles (dummies, stubs, fakes, and mocks)
- Microsoft Fakes framework (stubs and shims)
- Comparing mocking frameworks
- Using moq .NET mocking framework
- Debugging and profiling slow-running unit tests
- Using IntelliTest with legacy code