Course duration

- 2 days

Course Benefits

- 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

Course Outline

1. Unit Testing in .NET
    1. What is (and isn't) a unit test
    2. Why write unit tests
    3. .NET unit testing frameworks
    4. MSTest V2, NUnit, xUnit
    5. The anatomy of a unit test
    6. Writing and running your first unit test
2. Unit Testing in Visual Studio
    1. Testing support in Visual Studio
    2. MSTest, NUnit, and xUnit test projects
    3. Test Explorer and other windows
    4. Writing and running unit tests in Visual Studio
    5. Managing a large number of tests and test results
    6. Organizing tests by grouping, filtering, and playlists
    7. Continuous testing in Visual Studio
3. Test-Driven Development (TDD)
    1. TDD overview and benefits
    2. Practicing TDD within Visual Studio
    3. Effectively refactoring code
    4. Working with legacy code
    5. Using CodeLens to support TDD and refactoring
4. Writing Good Unit Tests
    1. Analyzing code coverage
    2. Using code coverage as a metric
    3. Parameterized (data-driven) unit tests
    4. DataRow, DynamicData, and DataSource attributes
    5. Concurrent testing using Live Unit Testing
    6. Concurrent testing using NCrunch
5. Testing Difficult Code
    1. The need to isolate code under test
    2. Doubles (dummies, stubs, fakes, and mocks)
    3. Microsoft Fakes framework (stubs and shims)
    4. Comparing mocking frameworks
    5. Using Rhino Mocks and NSubstitute frameworks
    6. Profiling slow running unit tests

7. Using IntelliTest with legacy code

Class Materials

Each student will receive a comprehensive set of materials, including course notes and all the class examples.

Class Prerequisites

Experience in the following *is required* for this ASP.NET class:

- Experience or familiarity with the C# language.
- Experience or familiarity with Visual Studio 2015, 2017, or 2019.
- Experience or familiarity with writing, debugging, and maintaining code.
- Experience or familiarity with Application Lifecycle Management basics.
- Experience or familiarity with your organization’s development lifecycle.
- Experience or familiarity with building a high-quality software product.