Unit Testing in Visual Studio 2017 (VS-114)

Course ID #: 1411-727-17-W

Hours: 14

Course Content

Course Description:

This Unit Testing in Visual Studio 2017 training course teaches attendees how to use Visual Studio 2017 to design, write, and run high-quality .NET unit tests. Students learn how to use 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 development lifecycle.

Note: This course can also be taught using Visual Studio 2015.

At Course Completion:

After competing this course, student will be able to:

- Work with .NET unit testing frameworks
- Write and run unit tests and managing test results.
- Practice Test-Driven Development (TDD)
- Write high-quality unit tests
- Use additional unit testing features found in Visual Studio
- Use tools and techniques for testing difficult code

Topics:

Introduction

Unit Testing in .NET

- The role of the developer
- Unit tests explained
- .NET unit testing frameworks
- MSTest, NUnit, xUnit.net, and others
- The anatomy of a unit test
- Writing your first unit test

Unit Testing in Visual Studio:

- Testing support in Visual Studio
- Test projects
- Test Explorer and other windows
- Unit testing in Visual Studio
- Running tests
- Managing test results
- Managing a large number of tests

Test-Driven Development (TDD)

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

Writing Good Unit Tests

- Know your code
- Path testing (i.e. sad path)
- Right BICEP
- Testing for expected exceptions
- Maintaining high-quality test code
- Unit test naming conventions (e.g. BDD)

- Organizing unit tests

Advanced Unit Testing in Visual Studio
- Code coverage
- Using code coverage as a metric
- Data-driven unit tests
- Continuous testing in Visual Studio
- Concurrent testing using Ncrunch

Testing Difficult Code
- The need to isolate code under test
- Doubles (dummies, stubs, fakes, and mocks)
- Microsoft Fakes framework (stubs and shims)
- Mocking frameworks (Rhino Mocks)
- Profiling slow running unit tests
- Using IntelliTest with legacy code

Conclusion