UNIT TESTING IN VISUAL STUDIO 2013

UTVS2013 | 2 Days

Overview

This two-day, instructor-led course provides students with the knowledge and skills to effectively use Visual Studio 2013 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 (TDD). This course also introduces other, popular unit testing tools and techniques, and demonstrates how they integrate with Visual Studio and your development lifecycle.

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, CI, refactoring, and how to test difficult code.

Prerequisites

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

- Building a high-quality software product
- Writing, debugging, and maintaining code
- The Visual C# language
- Visual Studio 2010, 2012, or 2013
- Their organization's development lifecycle

At Course Completion

This course teaches students how to effectively design, write, and run high-quality unit tests using Visual Studio. At course completion, attendees will have had exposure to:

- Why unit tests are critical to software quality
- How unit tests and integration tests differ
- Popular unit testing frameworks
- The anatomy of a unit test
- The 3A pattern (Arrange, Act, Assert)
- Assertions and expected exceptions
- Test class inheritance
- Testing private methods
- Visual Studio test projects
- Visual Studio test windows and tools
- How basic and standard unit tests differ
- How and when to use categories and lists
- How and when to use ordered tests
- Running tests and managing test results
- The importance of Test-Driven Development
- Implementing TDD in Visual Studio
- How to effectively refactor within TDD
- Why and how to refactor legacy code
- Happy path vs. sad path testing
- Testing boundary conditions
- Organizing tests and test assemblies
- Behavior-Driven Development principles
- How to use data-driven unit tests
- Why and how to calculate code coverage
- Why and how to use test impact analysis
- Team Foundation Server support for testing
- Team Foundation Build support for testing
- Why and how to use continuous integration
- Using dummies, fakes, stubs, and mocks
- Why and how to use Microsoft Fakes
- Why and how to use Rhino Mocks
- Why and how to use Microsoft Code Digger

Course Designer

This course was designed by Richard Hundhausen, a Visual Studio ALM MVP, Microsoft Regional Director, and an experienced software developer and trainer. For more information about his company, or to see other courses that they offer, visit www.accentient.com.

UNIT TESTING IN VISUAL STUDIO 2013

UTVS2013 | 2 Days

Modules

	Lessons	Hands-On
UNIT TESTING IN .NET This module introduces the concepts of unit testing and how it is supported by various unit testing frameworks and tools for .NET, including MSTest and NUnit. The anatomy of a unit test is also detailed in this module.	• The role of the developer • Unit tests explained • .NET unit testing frameworks (MSTest.exe and others) • Writing unit tests	• Setup the learning environment • Create and run a unit test • Create and run an integration test • Migrate NUnit tests to MSTest (optional)
UNIT TESTING IN VISUAL STUDIO This module introduces Visual Studio test projects, the testing windows, and the functionality for effectively writing and running unit tests and managing test results.	• Testing support in Visual Studio • Test projects • Test Explorer and other windows • Unit testing in Visual Studio • Managing a large number of tests • Running tests • Managing test results	• Review existing codebase and tests • Execute tests in various ways • Manage a large number of tests • Execute tests in sequence • Manage tests using test lists • Manage test results
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 a part of this module.	• TDD overview, benefits, and common objections • Implementing TDD in Visual Studio • Refactoring • Legacy Code	• The Bowling Kata • TDD and Refactoring
WRITING GOOD UNIT TESTS Just knowing how to write unit tests and being disciplined in TDD is not enough. This module will demonstrate several other practices for ensuring that you write high-quality unit tests.	• Know your code • Path testing (i.e. sad path) • Right BICEP • Testing for expected exceptions • Maintaining high-quality test code • Behavior-Driven Development (BDD) • BDD naming conventions • Organizing test libraries (assemblies)	• Handle bad input values • Refactor unit tests • Add a sad path test • The challenge!
ADVANCED UNIT TESTING TOOLS This module examines additional unit testing features found in Visual Studio, including code coverage, data-driven unit tests, and test impact analysis. In addition, many unit-testing specific features in Team Foundation Server and Team Foundation Build will be part of the discussion.	• Code coverage • Data-driven unit tests • CodeLens (Ultimate edition) • Test impact analysis • Team Foundation Server • Team Foundation Build • Build verification tests (BVTs) • Gated check-ins • Continuous Integration (CI)	• Use a data-driven unit test • Calculate code coverage • Use test impact analysis • Using CodeLens (optional)
TESTING DIFFICULT CODE This module introduces some tools and techniques for testing difficult code, such as code that can't be tested without being hosted in another environment (i.e.ASP.NET, SharePoint, etc.)	• The need to isolate code under test • Doubles (dummies, stubs, fakes, and mocks) • Mocking frameworks (Rhino Mocks) • Fakes framework (Ultimate edition) • Microsoft Code Digger	• Explore Microsoft Fakes framework • Explore Rhino Mocks (optional) • Explore Microsoft Pex (optional)