KOENIG
step forward

Unit Testing in Visual Studio

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

KOENIG
step forward

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, data-driven 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