Test Driven Development in C#

Duration:	3 days
Type:	intermediate

Description

This course introduces C# developers to the tools and techniques involved in Test Driven Development. The course is workshop based, with delegates spending the majority of their time applying the techniques they have learnt to sample applications. As the course progresses the following tools are incrementally included: NUnit, NMock, ReSharper, Unity, Spring .NET, WatiN and Selenium. Other tools can be covered as required.

Prerequisites

Delegates should be experienced and confident C# developers. Prior experience with DI containers and ASP .NET MVC is helpful but not essential.

List of Modules

The Business Case for Unit Testing and TDD

- Classifying the different types of testing
- What role do developers play in testing?
- Problems with ad-hoc unit tests
- Advantages of systematic unit testing
- Test Driven verses Behavior Driven Development

The Role of TDD in Modern Software Design

- Is there such a thing as Test Driven Design?
- The role of TDD in traditional processes
- The role of TDD in Agile processes
- Situations where TDD cannot succeed

Key Activities in TDD

- Defining your intent through tests
- Writing just enough code to pass
- Adding tests and refining the code
- Testing up to the point of boredom
- Triangulating on hard problems
- Refactoring the code and tests
- Building a suite of test cases

Unit Testing with NUnit and TFS

- Declaring test classes and methods
- Setting up and destroying resources
- Testing error handling behavior
- Running tests from build scripts

Unit Testing and Mock Objects

- Real world classes have dependencies
- Encapsulated dependencies prevent testing
- Using dependency injection to enable testing
- Distinguishing the different types of Test Double
- Understanding Stubs, Spys, Mocks and Fakes
- Writing your own mocking tools via CodeDOM

Mocking with NMock

- Creating simple Mock Objects
- Adding expectations to mock objects
- Configuring return values and exceptions
- Declaring custom Matchers and Actions

Refactoring in Depth

- Refactoring as the 'second hat'
- Refactoring is essential to TDD
- Support for refactoring in Visual Studio
- Enhanced refactoring support in ReSharper
- Refactoring to keep the code alive
- Detecting smells in code and tests
- The most productive refactorings

Integration Testing and Dependency Injection

- Moving from Unit Tests to Integration Tests
- The importance of DI Containers to Integration Tests
- Using the Unity Dependency Injection Container
- Using the Spring .NET Dependency Injection Container

TDD and Web Applications

- How ASP .NET MVC enables testable Web Applications
- Applying TDD concepts to MVC based web development
- Integration testing Web Applications using WatiN and Selenium