

# Test-Driven Development in C#

## Unit Testing and Refactoring for Agile Software Development

The *Test-Driven Development in C#* course presents a number of modern practices for developing code based on an iterative and incremental development lifecycle. Agility and predictability are two qualities often missing from software development. A test-driven approach, in which design is grown and code delivered incrementally according to functionality or risk, forms the basis of the construction phase of an iterative and incremental development. The use of unit testing provides confidence in existing code and supports the ability to refactor code as development unfolds.

The course is intended as a practical course: the best way to appreciate how test-driven development works and what it feels like is to do it in practice, making sense of the principles it embodies. In this form the course is based on lecture material, demonstration, discussion and hands-on labs. However, the course can also be run as a seminar without hands-on labs, which puts more emphasis on understanding the principles through the demonstrations.

### Objectives

- Appreciate the benefits of a continuous and iterative approach to design and delivery
- Recognise the purpose and practice of refactoring in keeping a system supple and adaptable
- Know how to build up a set of unit tests in NUnit 2
- Understand the consequences of dependency management on testing and code quality

### Audience

The course is suitable for software developers experienced in C# and familiar with object-oriented principles and practices. Any previous exposure to NUnit or agile development concepts is beneficial but not essential.

### Content

**Agile Development Microprocess** Traditional versus agile development processes · Iterative and incremental development · Informal and continuous design · The role of refactoring · Refactoring versus other code changes · Extreme Programming · Test-Driven Development

**Testing in Principle** Traditional view and reality of testing · Driving development through testing · Testing early, often and automatically · Testing versus debugging · White-box versus black-box testing · Functional versus operational testing

**Basic Unit Testing in Practice** Test plans versus test code · Use of *Debug.Assert* · Testing at the interface · Testing the simplest things first · Testing incrementally · Testing correctness of failure

**Overview of NUnit** NUnit and the xUnit family · Test fixtures and test methods · The role of attributes in NUnit's design · Assertion methods · Testing correctness of exceptions · Defining common fixture code

**Test-Writing Techniques** Red, green, refactor · None to one to many · Faking it · Telling the truth · Isolated and short tests · Refactor common fixture code · Declare, prepare, assert · Test by method, state or scenario · Custom assertions

**Common Refactorings** Renaming variables, methods, classes and packages · Restructuring class hierarchies by extracting interfaces, superclasses and subclasses · Partitioning classes by extracting classes and methods · Changing private representation

**Decoupling Techniques** Unmanaged dependencies · Test-driven decoupling · Layering · Reorganising packages · Eliminating cyclic dependencies · Mock objects · Eliminating Singletons, *statics* and other globals · Testing I/O

### Additional Details

**Duration** 2 days

**Setup** Projection facilities for a laptop · Whiteboard or flip chart · Workstations (one per pair of developers) with suitable development environment installed

**Contact** Kevlin Henney · kevin@curbralan.com · Curbralan Limited · +44 117 942 2990