

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 adaptive 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, exercises and hands-on labs.

Objectives

- Appreciate the benefits of testing as a design tool and not just a defect reduction technique
- 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
- 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

Programmer Testing Evidence of care · Testing viewpoint · Pragmatic testing · Automation · Bug pathology · Qualitative and constructive

Good Unit Tests Test quality · Good unit tests (GUTs) · Not-so-good unit tests · Fine-grained tests · Behavioural tests · Functional versus operational testing · Black-box tests

Overview of NUnit NUnit and the xUnit family · Tests in NUnit · NUnit assertions · Testing thrown exceptions · Other NUnit features · Organising NUnit tests

Testing Approach Testing sensibility: passive, reactive, active · Plain Ol' Unit Testing (POUT) · Defect-Driven Testing (DDT) · Test-Driven Development · Key TDD practices · The test-first cycle · Definition of done

More GUTs Cohesive and focused tests · Propositional test names · Example-based tests · Choosing example data · Quality of failure · What to include and exclude · Short test cases · Single level of abstraction · Anatomy of a test case (Given–When–Then)

Listening to Your Tests Feedback from testing · Reasons testing can be hard · Technical debt · Classifying and reacting to technical debt · White-box testing issues · Coverage

Refactoring Kinds of changes to code · Code smells · Dimensions of change · Elements of refactoring · Some common refactorings · Refactoring motivation and applicability

Test-Driven Decoupling Unit testability and coupling · Unmanaged dependencies · External dependencies · Unit versus integration tests · Characterising testability · Singletons and *statics* · Test doubles · Mock object frameworks

Additional Details

Duration 2 days (can be extended to 3 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