

Applying Design Patterns in C#

Duration:	3 days
Type:	intermediate

Description

This course is designed for developers who want to extend their architectural skills using design patterns and related techniques. It is heavily interactive with delegates spending the majority of their time creating pattern based solutions to real world problems.

Each pattern is introduced in terms of its abstract structure (via UML diagrams), its benefits and drawbacks, sample implementations in C# and practical examples of how it can be used to simplify software development.

The course makes extensive use of the .NET framework libraries as examples of pattern based architecture. In addition it covers C# specific variations on, and alternatives to, the standard 'Gang of 4' patterns.

Prerequisites

Delegates should be experienced C# developers familiar with UML Sequence and Class diagrams.

List of Modules

Introduction to Patterns

- What is a design pattern?
- The evolution of design patterns
- Misconceptions about design patterns
- The dangers of becoming 'pattern happy'
- Distinguishing between patterns, idioms and refactorings
- Using refactorings to introduce patterns incrementally
- Using patterns to create an object oriented architecture

Revision of Core OO and SOLID

- Abstraction, encapsulation, inheritance and polymorphism
- The importance of the SOLID principles in OO development
- The Single Responsibility and Open / Closed principles
- The Liskov Substitution and Interface Segregation principles
- The Dependency Inversion principle and Inversion of Control

MVC and MVP

- Why classes take on different roles in a layered architecture
- How MVC emerged in Smalltalk and has been universally applied
- ASP .NET MVC as an example of an MVC based .NET framework
- The evolution of the Model View Presenter (MVP) pattern
- Distinguishing between the MVC and MVP patterns
- Examples of MVP in ASP .NET and WPF

The Little Language Pattern

- When creating your own language is appropriate
- SQL, Regular Expressions and XPath as small languages
- LINQ as an example of the Little Language Pattern
- Little Language and Domain Specific Languages (DSL)
- The role of Little Language in Software Factories

Proxy

- Adding services to objects by intercepting messages
- How proxies are using in Windows Communication Foundation (WCF)
- Using the .NET interception framework to add proxying to your code
- Creating proxy classes using context bound objects
- Creating proxy classes using *Reflection.Emit*
- NMock as a practical example of dynamic proxies

Factory

- Advantages of separating clients from object creation
- Comparing the Factory Method and Abstract Factory Patterns
- Extending the Factory Pattern into Dependency Injection
- Open Source Dependency Injection containers available for .NET
- The Managed Extensibility Framework (MSF) in Visual Studio 2010

Composite

- Modelling nested whole-part relationships in OO
- Examples of Composite in XML and GUI libraries

Singleton

- Why ensure a class only has a single instance?
- General problems implementing Singleton objects
- Language specific problems with Singletons (Java and C++)
- Scala as an example of linguistic support for Singletons
- Options for creating Singleton classes in C# and VB .NET

Strategy

- Creating class hierarchies to represent algorithms
- Separating a class from a changing or complex algorithm
- How Strategy is used in the .NET Collections Libraries

Command

- Similarities between Strategy and Command
- Using Command objects to simplify event dispatching
- Incrementally refactoring code to introduce Command

Template

- Using polymorphism to customize algorithms
- Similarities between Template and Factory Method
- How Template is used when writing ASP .NET Controls

Decorator and Adapter

- Using composition to layer extra functionality
- How Decorator is used in the .NET I/O libraries
- Distinguishing between Adapter and Decorator
- Using adapter to communicate with COM components

Iterator

- Accessing an aggregate object without knowing its representation
- How Iterator is used in the STL, the Java and .NET collections
- Adding iterator support to your own collections

Observer

- Informing interested objects of state changes
- Benefits and dangers in implementing Observer
- The Observer Pattern, delegates and events in C#

Visitor

- Simplifying class design by modelling operations as visitors
- Adding support for Visitor to existing collections of objects
- Using Visitor to add reporting and logging behaviour
- Using Visitor in the design of a mock objects generator

State

- Benefits of the State Pattern over subclassing
- Modelling objects with complex internal state transitions
- Different approaches to implementing state transitions
- Automatically generating state machines

Basic Threading Patterns

- Implementing the Active Object Pattern in Java, C# and C++
- The Thread Pool Pattern and how it is used in .NET
- Introducing the IOU (Asynchronous Completion Token) Pattern
- How the IOU Pattern is used in .NET (the Async Pattern)

Parallel Programming Patterns

- Overview of the new threading features in .NET 4 (VS2010)
- Parallel loops and running LINQ queries in parallel (PLINQ)
- Using the Fork / Join and MapReduce Patterns effectively