

.NET Best Practices and Design Patterns: Hands-On - 4 Days

Building Successful Applications with Proven Techniques

Course 511 Overview

- You Will Learn How To**
- Implement proven methods to build adaptable, reliable and efficient .NET Web and smart client applications
 - Solve complex programming problems with industry-standard design patterns
 - Deliver bug-free code using test-driven development in Visual Studio
 - Create layered architectures for reusability and avoiding functional overlap
 - Apply best practices for improved class design
 - Simplify and automate repetitive tasks
- Course Benefits** Knowledge of the .NET languages and libraries alone is not enough to develop robust applications. Proven design patterns and best practices, distilled from the wisdom of experts, support you in building applications upon a solid foundation. This course provides the skills needed to solve real-world software development problems and deliver fast, reliable applications.
- Who Should Attend** Programmers, system architects and anyone developing .NET applications. Programming experience at the level of Course 419, "C# Programming," or Course 503, "Visual Basic Programming for .NET," is assumed.
- Hands-On Training** You gain experience implementing .NET best practices and design patterns. Exercises, completed in VB or C#, include:
- Reconciling diverse processing problems with the Strategy pattern
 - Simplifying a complex system with the Facade pattern
 - Exploiting Microsoft Entity Framework for data access and updates
 - Creating a Business Domain Object Model
 - Building automated test cases
 - Structuring a testable user interface with the Model View Controller pattern
 - Capturing and reusing tests within Visual Studio

.NET Best Practices and Design Patterns: Hands-On - 4 Days

Building Successful Applications with Proven Techniques

Course 511 Outline

Introduction

- Coding with best practices
- Simplifying software with design patterns

Simplifying Complex Programming with Proven Design Patterns

Applying simple interfaces to intricate algorithms

- Unifying interfaces from subsystems for ease of use and reusability
- Implementing the Facade pattern

Varying functionality by programming to interfaces

- Enhancing the adaptability and flexibility of your application
- Exploiting the Strategy pattern

Extending object behaviour dynamically

- Increasing functionality without impacting existing code
- Composing objects with the Decorator pattern

Achieving reuse and flexibility

- Eliminating code duplication by outlining a basis for an algorithm
- Employing the Template Method pattern

Interfacing incompatible classes

- Transforming an interface to add value to existing code
- Harnessing the Adapter pattern

Applying Test-Driven Development Techniques

Automating unit testing

- Shortening development cycles with automated tests
- Improving quality with consistent test coverage
- Eliminating regression errors with reusable tests

Integrating testing and coding

- Applying the test-first programming practice to drive code design
- Generating immediate green-light feedback for increased code quality and short feature development cycles
- Organizing, coordinating and running test cases with Visual Studio

Architecting a Layered Application

Designing the application architecture

- Layering architectures for reusability, durability and scalability
- Accessing data from the business tier
- Decoupling object creation with the Factory pattern
- Preserving object identity with the Identity Map pattern

Programming application tiers

- Structuring a Web or rich client application with the Observer pattern
- Organizing state-rich applications with the State pattern
- Restructuring database tables without impacting application code

Modeling a business area

- Liberating rich Business Object Models from database structures using the Domain Model pattern
- Mapping rich Business Objects to database tables with the Data Mapper pattern
- Collapsing Business Object hierarchies with the Inheritance Mapper pattern
- Exploiting declarative programming in Microsoft Entity Framework to implement Data Mapping classes

Applying Best Practices

Organizing and implementing business logic

- Applying domain logic patterns in the middle tier
- Extending Entity types with partial classes
- Manipulating groups of Entities with business logic classes

Best practices in class design

- Guarding against rigidity with the Open/Closed Principle
- Extracting new classes with the Single Responsibility Principle
- Effective use of inheritance

Automating Repetitive Tasks

Easing data access code

- Reducing database access code by exploiting Entity Navigation Properties
- Eliminating database update code with Entity change tracking

Enhancing application code

- Automating design reviews with FxCop and Visual Studio Analysis
- Eliminating code duplication through refactoring to design patterns