SE013CS - *Refactoring to Patterns using C#*

DURATION: 3 Days; Instructor-led

WHAT YOU WILL LEARN

Often developers who leaned to the basic Software Design Patterns find hard to apply them into development.

The course provides step-by-step instructions on how to improve your code through the methodical introduction of appropriate patterns, but more so because it teaches the principles that underlie the design patterns implemented.

In GoF Design Patterns claimed that it is targeting for refactorings. This course shows that how to realize the claim by deepen your understanding of both refactoring and design patterns."

AUDIENCE

This course should be useful for novice and expert designers alike. It is an invaluable training for the refactoring practitioner

PREREQUISITES:

- Basic knowledge and skill in C#.
- At least 3 years of experience in software development and preferably with basic understanding of software design patterns.

METHODOLOGY

This program will be conducted with interactive lectures, PowerPoint presentations, discussions and practical exercises

COURSE OUTLINES

Module 1 - Introduction

- The Patterns Panacea
- Under-Engineering
- Test-Driven Development and Continuous Refactoring
- Refactoring and Patterns
- Evolutionary Design

Module 2 – Refactoring

- What Is Refactoring?
- The motivations to refactor.
- Human-Readable Code
- Keeping It Clean
- Small Steps
- Design Debt
- Evolving a New Architecture
- Composite and Test-Driven Refactorings
- The Benefits of Composite Refactorings
- Refactoring Tools

Module 3 – Patterns

- What Is a Pattern?
- Very brief introduction to the GoF Patterns
- There Are Many Ways to Implement a Pattern
- Refactoring to, towards, and away from Patterns
- Do Patterns Make Code More Complex?
- Pattern Knowledge
- Up-Front Design with Patterns

Module 4 – The Indicators

- Duplicated Code
- Long Method
- Conditional Complexity
- Primitive Obsession
- Indecent Exposure
- Solution Sprawl
- Alternative Classes with Different Interfaces
- Lazy Class
- Large Class
- Switch Statements
- Combinatorial Explosion
- Oddball Solution

Module 5 - A Catalog of Refactorings to Patterns

- Format of the Refactorings
- Projects Referenced in This Catalog
- A Starting Point
- A Study Sequence

Module 6 – Creation

- Replace Constructors with Creation Methods
- Move Creation Knowledge to Factory
- Encapsulate Classes with Factory
- Introduce Polymorphic Creation with Factory Method
- Encapsulate Composite with Builder
- Inline Singleton

Module 7 - Simplification

- Compose Method
- Replace Conditional Logic with Strategy
- Move Embellishment to Decorator
- Replace State-Altering Conditionals with State
- Replace Implicit Tree with Composite
- Replace Conditional Dispatcher with Command

Module 8 - Generalization

- Form Template Method
- Extract Composite
- Replace One-Many Distinctions with Composite
- Replace Hard-Coded Notifications with Observer
- Unify Interfaces with Adapter
- Extract Adapter
- Replace Implicit Language with Interpreter

Module 9 - Protection

- Replace Type Code with Class
- Limit Instantiation with Singleton
- Introduce Null Object

Module 10 - Accumulation

- Move Accumulation to Collecting Parameter
- Move Accumulation to Visitor

Module 11 - Utilities

- Chain Constructors
- Unify Interfaces
- Extract Parameter