



CCRT - Version: 1
01 May 2021

Clean Code and Refactoring Techniques



Clean Code and Refactoring Techniques

CCRT - Version: 1

 2 days Course

Description:

Writing clean code is an essential skill for every developer. It saves a lot of time in debugging, maintenance and bug fixing down the road. However, if we also want to be able to quickly respond to changing requirements and still keep our code clean, we must also know how to refactor it efficiently. In this course we'll learn and practice some basic ideas of clean code and refactoring techniques, with the goal of improving these skills.

Intended audience:

Professional developers (examples and exercises are in C#, but most content is relevant also for Java developers), architects, technical team leads

Prerequisites:

At least 1 year experience developing software using C# or another object-oriented language

Understanding of Object-Oriented concepts.

Objectives:

Understand the importance of clean code and refactoring.

Learn and practice some basic practices for clean-code.

Understand the SOLID principles and the 4 rules of simple design.

Understand the best practices for performing refactoring.

Understand the importance of unit tests and TDD for clean code.

Topics:

° The Value of Clean Code and Refactoring

Refactoring Guidelines

- Small steps
- Keep it working
- Use //TODOs and [Obsolete]

Basic Clean Code Practices

- Readability: Naming, spacing
- Number of arguments
- Method length
- Cyclomatic Complexity
- Exception handling
- Poka-Yoke
- Exercise

Modularity, Reuse and Design

- Avoid static state
- Law of Demeter
- Cohesion and Coupling
- Polymorphism and function pointers
- The SOLID principles
- 4 rules of simple design

Specific Refactoring Techniques Using Resharper

- Extract/Inline methods
- Extract/Inline variables, fields and parameters

- Change method signature
- Extract super class/interface
- Pull members up/Push members down
- Move to another type
- Safe delete

Code Analysis Using Resharper

- Go to base/derived/declaration/implementation
- Analyze type hierarchy
- Analyze project references
- Call tracking
- Value tracking
- Dependencies diagram

Introduction to Unit Tests

- Unit tests structure
- Mock objects
- Limitations of unit tests

Introduction to TDD

- The way TDD works
- Benefits of TDD
- TDD and legacy code

Summary: Where to go next?

- Practice
- Pair programming / Code review
- Mentoring