Refactoring

This course serves as an indispensable guide to enhancing your codebase's quality and maintainability.

The course begins by helping you get to grips with refactoring fundamentals, including cultivating good coding habits and identifying red flags. You'll explore testing methodologies, essential refactoring techniques, and metaprogramming, as well as designing good architecture. The modules clearly explain how to refactor and improve your code using real-world examples and proven techniques. Part two equips you with the ability to recognize code smells, prioritize tasks, and employ automated refactoring tools, test frameworks, and code analysis tools. You'll discover best practices to ensure efficient code improvement so that you can navigate complexities with ease.

By the end of this course, you'll be able to avoid unproductive programming or architecting, detect red flags, and propose changes to improve the maintainability of your codebase.

What You Will Learn:

- Recognize and address common issues in your code.
- Find out how to determine which improvements are most important.
- Implement techniques such as using polymorphism instead of conditions.
- Efficiently leverage tools for streamlining refactoring processes.
- Enhance code reliability through effective testing practices.
- Develop the skills needed for clean and readable code presentation.
- Apply best practices for a more efficient coding workflow.

Prerequisites

Participants should have:

- Basic knowledge of Object-Oriented Programming
- Basic Programming knowledge and skill in the language in used. This course is one can run for any one of the following languages:
    - C#
    - C++
    - Java
    - Python

Target Participants

This course is for C/C++/Java/Python developers, software architects, and technical leads looking for a comprehensive course to advancing their skills in software design and refactoring. The course is ideal for experienced programming enthusiasts, quality assurance engineers, and codebase maintainers as it provides practical insights, real-world examples, and essential patterns. Development managers who want to foster clean coding practices by using best practices for efficient workflows will also find this course useful.

Course Outline

PART-1: Introduction to Refactoring

Module 1: What is Refactoring?

- What do we mean by refactoring?
- Refactoring and clean code
- Misconceptions about refactoring
- Why you should consider refactoring
- Improving the design of your software
- Maintainability and scalability
- Understanding, avoiding, and fixing bugs
- Faster development
- When you should refactor
- The "Rule of Three"
- Preparatory refactoring for smooth feature implementation
- Refactoring for bug fixing
- Comprehension refactoring
- The "Boy Scout Rule"
- Planned refactoring
- Long-term refactoring and "Branch by Abstraction"
- Refactoring in a code review
- What you should refactor
- Impact analysis
- Risk assessment
- Value estimation
- Prioritization matrix
- Refactor or rewrite?

Module 2: Good Coding Habits

- Characteristics of good code
- Good code is readable
- Good code is reliable
- Good code is hard to misuse
- Good code is modular
- Good code is reusable
- Clean Code
- Why Clean Code?
- Some principles of Clean Code
- Write SOLID code
- Single responsibility principle
- Open-closed principle
- Liskov substitution principle
- Interface segregation principle
- Dependency inversion principle
- Side effects and mutability
- The builder pattern
- Causes of bad code
- Deadlines
- The Broken Window theory
- No code review process
- Insufficient domain or technical knowledge

PART-2: Essence of Refactoring and Good Code

Module 3: Code Smells

- Duplicated code
- Long methods
- Replace temporary variables with query methods
- Parameter object
- Replace a function or method with a command
- Decompose conditionals
- Split loops
- Large classes
- Switches
- Primitive Obsession
- Middle man
- Message chains

- Feature envy methods
- Divergent change
- Shotgun Surgery
- God object

Module 4: Testing

- Why you should test (often)
- Identifying and fixing defects
- Ensuring quality and reliability
- Customer satisfaction
- Compliance and standards
- Security
- Integration and compatibility
- Confidence and peace of mind
- Unit testing
- Test framework
- Integration testing
- Mockito
- Contract testing
- Be safe – checking and improving your test coverage
- What is test coverage?
- Test-driven development
- Advantages of TDD

Module 5: Refactoring Techniques

- Writing better methods
- Extract Method
- Inline Method
- Extract and inline variables
- Combining a function into a class
- Moving features between objects
- Moving a method or field
- Moving statements into/from methods
- Hiding delegates
- Removing dead code
- Organizing data
- Field encapsulation
- Replacing primitives with objects
- Replacing type code with subclasses

- Simplifying conditional logic
- Returning a special case instead of null
- Using polymorphism instead of conditions
- Removing duplicated conditions
- Guard clauses
- Simplifying method calls
- Avoiding side effects
- Removing setter methods
- Using generalization
- Pull up field
- Push down field
- Pull up method
- Push down method
- Template method
- Using enums instead of constants

Module 6: Static and Dynamic Analysis

- What is static analysis?
- Code errors or bad practices
- Security
- Cyclomatic complexity
- Exception (mis)handling
- Automated static analysis tools
- What is dynamic analysis?
- Debugging
- Profiling
- Fuzzing
- Symbolic execution
- Taint tracking

Suggested Course Duration:

Intensive Bootcamp (3-4 Days, Full-Time)

- Suitable for professionals who want rapid upskilling
- Daily sessions of 6-8 hours
- Fast paced with hands-on projects