

Kod szkolenia: **C/REFACT**

Tytuł szkolenia: **Refactoring techniques in improving code quality and implementation of design patterns in C++**

Dni: 3

Opis:

Course Description

This training course presents refactoring techniques and their usage in improving code quality and implementation of design patterns. It provides fundamentals on assessment of code quality, including the catalog of common code smells and antipatterns. The key part of the training course is a workshop with practical exercises on refactoring techniques and Gang-of-Four design patterns.

Target Audience

Professional developers who want to learn about refactoring and implementation of design patterns in C++.

Course Goals

After the course, the student will be able:

- to assess code quality
- to point and name code smells and antipatterns as well as to discuss their negative impact on code quality
- to understand and use various refactoring techniques
- to understand the usage contexts of various design patterns along with their implementation

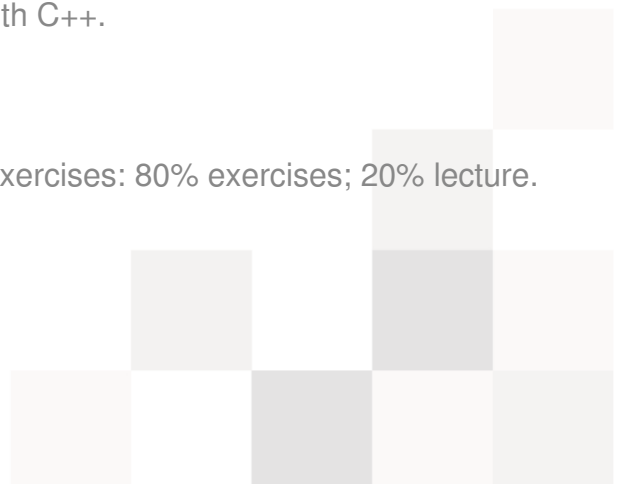
Pre-requisites

Good knowledge of object-oriented programming with C++.

Parameters

3*8 hours (3*7 net hours) of lectures and practical exercises: 80% exercises; 20% lecture.

Program szkolenia:



1. Introduction

- I. Object-oriented paradigms
- II. What makes good software
- III. Design patterns and principles
- IV. GRASP, SOLID and GoF

2. Code quality

- I. How to measure code quality?
- II. Code smells

Duplicate Code, Long Parameter List, Long Methods, Large Classes, god class, Dependency on Implementation, Meaningless Names, Data Clumps, Primitive Obsession, Switch Statements, Lazy Class, Speculative Generality, and other code smells

III. Antipatterns

Cut-and-Paste Programming, Spaghetti Code, Programming to Implementation, Tight coupling, Golden Hammer, Poltergeist, Boat Anchor, Dead End, Ambiguous Viewpoint, Lava Flow, Mushroom Management, and other antipatterns

IV. Improving code quality

Refactoring process
Unit tests for components under refactoring
Clean code
Encapsulation
Composition over inheritance
Programming to interfaces
SOLID:

V. Technology debt

3. Refactoring techniques

I. Introduction

II. Composing methods

Introduce Explaining Variable
Split Temporary Variable
Replace Template with Query, Inline Temp, Inline Method i Extract Method
Remove Assignments to Parameters
Substitute Algorithm

III. Simplifying method calls

Replace Constructor with Factory Method, ErrorCode with Exception, Exception with Test, Parameter with Explicit Methods, Parameter with Method,
Introduce Parameter Object
Encapsulate Downcast

IV. Moving features between objects

Move Method i Move Field
Extract Class
Inline Class
Hide Delegate



- Remove Middle Man
- Introduce Foreign Method
- Introduce Local Extension

V. Organizing data

- Replace Array with Object, Data Value with Object, Magic Number with Symbolic Constant, Record with Data Class, Subclass with Fields, Type Code with Class, Type Code with Strategy/State, Type Code with Subclasses
- Encapsulate Collection, Encapsulate Field
- Change Bidirectional Association to Unidirectional, Unidirectional Association to Bidirectional, Reference to Value, Value to Reference

VI. Simplifying conditional expressions

- Decompose Conditional Expressions, Consolidate Conditional Expressions i Consolidate Duplicate Conditional Expressions
- Remove Control Flag
- Replace Nested Conditional with Guard Clauses
- Replace Conditional with Polymorphism
- Null Object

VII. Generalizations

- Pull Up Constructor Body, Field, Method
- Push Down Field, Method
- Collapse Hierarchy
- Extract Interface, Subclass, Superclass
- Replace Delegation with Inheritance, Inheritance with Delegation

4. GoF design patterns

- I. When and how to use design patterns

- II. Creational design patterns

 - Builder, Prototype, Factory Method, Abstract Factory, Singleton

- III. Structural design patterns

 - Facade, Proxy, Composite, Adapter, Decorator, Bridge

- IV. Behavioral design patterns

 - Command, Observer, State, Strategy, Chain of Responsibility, Mediator, Visitor, Template Method

5. Summary

