+44 7989 401397
info@olsensoft.com

# C++ Design and Best Practices (3 days)

## Course overview

C++ is an exquisitely rich language, packed with fascinating language features and extensive library capabilities. Learning the syntax and STL nuances is a task in itself.

This course goes beyond the syntax and library to focus on C++ design principles and best practices. The course describes how to design C++ systems so that they are modular, maintainable, extensible, and pluggable. We take a detailed look at the SOLID principles and see how to apply them using modern C++ techniques. We also lift the lid on many design patterns, implementation patterns and C++ idioms and see how they remain relevant in C++ today.

## What you'll learn

- Understanding the SOLID principles and how to apply them
- How to design for modularity, maintainability, extensibility, and pluggability
- How to design object relationships and class relationships effectively
- How to make use of static typing via templates and concepts
- How to apply design patterns using modern C++ techniques
- C++ coding best practices

## Prerequisites

- Solid programming experience in C++

## Course details

- Getting Started: The Importance of Software Design; The Art of Software Design; OO Modelling
- Designing for Change: The Single Responsibility Principle (SRP); SRP Example; The DRY Principle
- Segregating Interfaces: Interfaces in C++; The Interface Segregation Principle (ISP); Applying the ISP to Template Types
- Designing for Extensibility: The Open-Closed Principle (OCP); The OCP and Inheritance; The OCP and Template Specialization
- Designing Object Relationships: Setting the Scene; Association; Composition; Using Smart Pointers Appropriately
- Managing Resources with RAII: Overview of RAII; Real-World RAII Examples; Managing Dynamic Objects with RAII
- Designing for Consistency [The Principle of Least Surprise]: The Liskov Substitution Principle (LSP); The LSP and Expectation Management; Covariance and Contravariance; The LSP and Static Typing

- Designing for Pluggability: Defining a Pluggable Hierarchy; Managing Dependencies; Organizing Projects and Libraries
- Introduction to Design Patterns: Essential Concepts; Design Pattern Classification; Anti-Patterns; Design Heuristics
- Creational Design Patterns: Factory Method Pattern; Abstract Factory Pattern; Prototype Pattern; Builder Pattern
- Structural Design Patterns: Composite Pattern; Decorator Pattern; Static Polymorphism via Templates; Object Adapter / Class Adapter Patterns; Bridge Pattern
- Behavioural Design Patterns: State Pattern; Command Pattern; Strategy Pattern; Visitor Pattern; Implementing the Visitor Pattern via **`std::variant`**