Building Solid Software in C and C++

Contents

1 Overview

This course introduces principles and practical techniques for building solid software in C and C++.

Participants in the course will:

- Learn fundamental principles of software design and clean code.
- Understand the importance of testing and how to write effective tests.
- Explore strategies for decoupling at design, compilation, linking, and runtime.
- Gain practical techniques for improving and refactoring legacy code.
- Discover how to apply object-oriented principles in C.

2 Module 1: The principles you need to know (and those you should forget)

Duration: 90 minutes

Target Audience: Programmers of C and C++, architects

Prerequisites: Knowledge of C, familiarity with Object Oriented Design and C++

Highlights:

In this module we will review the most essential principles of software design.

Building on these, we will review the three pillars of OOD. Encapsulation, inheritance and polymorphism. Do we actually need them and why?

We will engage in an open discussion and the conclusions will be demonstrated with code examples. The examples will demonstrate common pitfalls, the techniques you can use to identify the problems, and how to fix them using simple C++.

3 Module 2: The importance of tests

Duration: 90 minutes

Target Audience: Programmers of C and C++, architects, technical leads

Prerequisites: Module 1

Highlights:

In this module we will learn why tests are so important. Verification is the obvious benefit, but there are many more.

We will describe different levels of tests: Unit Tests, Integration Tests, System Tests. Everyone does System Testing (hopefully) but is that enough?

Building on Module 1 we will learn what makes code testable and we will consider the benefits of Dependency Injection, Mocks and Stubs.

It is all actually much simpler than it may sound, and I will demonstrate with code examples to prove it.

4 Module 3: Applying decoupling to design, compilation, link and run time

Duration: 90 minutes

Target Audience: Programmers of C and C++

Prerequisites: Module 1 and an understanding of the build process for C/C++ projects

Highlights:

One of the conclusions from Modules 1 and 2 is that by decoupling the pieces of our software we gain testability, maintainability and extensibility.

So decoupling is essential.

This module covers the techniques we can use to achieve decoupling at different stages of the development cycle: design, compilation, linking and even at run time.

The examples will demonstrate some of the simple techniques that we can use: Programming by interface, header organization, the pimpl idiom and interfaces. We will also learn about shared objects, dynamic linking and dynamic loading.

The last example will use shared objects to demonstrate how decoupling enables partial software updates in the field and even the modification of the system at run time using plugins.

5 Module 4: How to improve legacy code

Duration: 90 minutes

Target Audience: Programmers of C and C++, team leaders

Prerequisites: Modules 1 and 2

Highlights:

Module 1 presents the principles, but how are those principles applied in practice?

In Module 2 we learned that one of the benefits of tests is that they reduce the risks involved in refactoring existing code.

But legacy code often has no tests - so refactoring may easily introduce bugs. Worse, code with no tests is often not testable, that is, we must refactor it in order to enable us to introduce tests.

This vicious cycle often leads to stagnation, and over time, bad code degrades further, becoming increasingly brittle and harder to maintain.

The recommended approach in such cases is to make incremental changes. Adding a test, doing some refactoring and then adding more tests.

This module is a coding demonstration, which begins with a working project which has issues. In an open discussion, we will identify the issues and then I will demonstrate the incremental approach to resolving them.

6 Module 5: Object Oriented Design and C

Duration: 90 minutes

Target Audience: C programmers and project leaders

Prerequisites: Module 1 and a familiarity with Object Oriented Design

Highlights:

In Module 1 we discussed the benefits of OOD. In particular, that it helps us enforce encapsulation and enables programming by interface.

But in some embedded environments, we do not have access to a C++ compiler, and we are stuck with C.

As Bjarne Stroustrup put it, C++ is a better C, but this is not to say we cannot emulate some capabilities in C.

In this module we will explore techniques for implementing Object Oriented Development in C. We will learn how to enforce the visibility of methods, implement interfaces, and work with callbacks that have contexts.

We will also learn about the interoperability of C and C++. This is important because it gives you the choice to upgrade parts of your project from C to C++ without affecting other parts.