Cloud Contraptions
TRAINING & CONSULTING

To discuss this course and customizations:
Call: +1 434-509-6890 or Email: sales@cloudcontraptions.com

Effective Modern C++

Class Duration

35 hours of live training delivered over 5 days.

Student Prerequisites

- Professional C++ programming experience
- Working knowledge of classes, pointers, and the standard library basics
- Familiarity with Git

Target Audience

Working C++ developers whose habits were formed before the modern standards, or who use modern features without full command of them. The course is built for teams maintaining significant C++ codebases who want to write safer, faster, more expressive code and modernize legacy code with confidence. Developers new to C++ should start with C++ and Object-Oriented Programming; this course assumes daily C++ fluency and goes deep on the idioms that separate contemporary C++ from the C++ of the 2000s.

Description

C++ has been reinvented across six standards, and this course covers the idioms that matter, up through C++26. Participants master the machinery of modern C++ (move semantics and forwarding, smart-pointer ownership design, type deduction, lambdas, concepts-constrained templates, ranges, and compile-time programming with constexpr and C++26 reflection), then apply it to the concurrency and error-handling patterns of production systems, including coroutines, parallel algorithms, standardized SIMD, and contracts. The course closes where most teams live: modernizing legacy code incrementally and safely. Every idiom is practiced against realistic, gnarly code, with agentic AI coding assistants working alongside participants for large-scale mechanical refactoring, test generation, and modernization proposals that participants learn to review critically.

To discuss this course and customizations:
Call: +1 434-509-6890 or Email: sales@cloudcontraptions.com

C++26 was finalized by the ISO committee in March 2026. Coverage of individual C++26 features reflects compiler support at delivery time.

Learning Outcomes

- Explain how move semantics, copy elision, and value categories actually work, and design APIs around them.
- Apply ownership-driven design with smart pointers, and identify code where raw pointers remain correct.
- Use auto, decltype, structured bindings, and CTAD with full awareness of their deduction rules.
- Write expressive callable code with lambdas, captures, and the modern function-object vocabulary.
- Constrain templates with concepts and migrate SFINAE-era code to cleaner equivalents.
- Compose data transformations with ranges, views, and projections.
- Move computation to compile time with constexpr, consteval, and C++26 reflection.
- Build correct concurrent code with threads, atomics, coroutines, and the parallel and SIMD facilities of the standard library.
- Drive agentic AI coding assistants through large-scale modernization refactors and review their output with expert judgment.

Training Materials

Comprehensive courseware is distributed online at the start of class. All students receive a downloadable MP4 recording of the training.

Software Requirements

Visual Studio 2026 (Windows) or VS Code with a current GCC or Clang toolchain (any platform), plus CMake and Git. Portions of the course covering C++26-specific features use GCC or Clang where MSVC support lags. Access to an agentic AI coding assistant is required; a free tier is sufficient.

Training Topics

The Modern C++ Landscape

- Six standards in fifteen years: what changed and why
- Compiler and toolchain support across GCC, Clang, MSVC
- The C++ Core Guidelines as a north star

- The memory-safety conversation and C++'s response
- Agentic AI coding assistants in modernization work

Value Semantics and Move Machinery

- Value categories: lvalues, rvalues, xvalues
- Move construction and move assignment
- Forwarding references and perfect forwarding
- Copy elision and RVO
- When moves don't happen: common pessimizations

Ownership and Resource Design

- unique_ptr as the default ownership vocabulary
- shared_ptr costs, aliasing, and control blocks
- weak_ptr and breaking reference cycles
- The rule of zero as a design goal
- Handles, views, and non-owning types

Type Deduction Mastery

- auto deduction rules and their surprises
- decltype and decltype(auto)
- Structured bindings
- Class template argument deduction
- When explicit types communicate better

Lambdas and Callables

- Capture semantics: by value, by reference, init-capture
- Generic and templated lambdas
- Lambdas vs. std::function vs. function_ref
- Stateful callables and algorithm customization

Templates and Concepts

- Concepts and requires clauses
- Designing good concepts
- Migrating SFINAE and enable_if code
- Variadic templates and fold expressions
- Explicit instantiation and compile-time hygiene

Ranges and Algorithms

- Ranges, views, and lazy composition

To discuss this course and customizations:
Call: +1 434-509-6890 or Email: sales@cloudcontraptions.com

- Projections and range adaptors
- Writing range-friendly interfaces
- Ranges vs. classic iterator algorithms

Compile-Time Programming

- constexpr and consteval in practice
- Compile-time data structures and algorithms
- C++26 reflection: the ^^ operator and metafunctions
- Generative uses of reflection: serialization, bindings

Concurrency and Parallelism

- Threads, jthread, and cooperative cancellation
- Atomics and the memory model, pragmatically
- Coroutines: generators and async tasks
- Parallel algorithms and execution policies
- Standardized SIMD in C++26

Error Handling, Contracts, and Safety

- Exception safety revisited for modern code
- std::optional, std::expected, and error-code design
- Contracts in C++26: pre, post, contract_assert
- Hardening: sanitizers and undefined-behavior hygiene
- Memory-safe coding guidelines and hardened library modes

Modernizing Legacy Code

- Incremental migration strategies that ship
- Modules: adoption paths for existing codebases
- clang-tidy modernize checks at scale
- AI-assisted refactoring campaigns and their review
- Measuring improvement: performance and defect metrics

