

Updated on 06/11/2025

Register

Clean Architecture .NET Core Training

3 days (21 hours)

Overview

Create a loosely coupled application with reverse dependencies!

Clean Architecture is a software architecture designed to keep code under control without having to organize it to prevent anyone from touching it after it has been published. The main concept of clean architecture is that the application code/logic must be written without any direct dependencies.

Whether you modify the database or the user interface, the core of the system (business/domain rules) must not be modified. This means that external dependencies are completely replaceable.

Clean architecture makes an application independent of any framework, database, or user interface. It is testable and well organized. This architecture consists of four distinct categories: Domain, Application, Infrastructure, and Presentation. This architecture is often called onion architecture, hexagonal architecture, or even port & adapt. The term "clean architecture" comes from [Uncle Bob](#).

Thanks to our Clean Architecture training in .NET Core, you will be able to create an architecture for developing testable and robust APIs that can be easily understood and reused by other developers.

In this training course, as in all of our training courses, we will use the [latest stable version](#) (.NET 9 with the new [C# 13 language](#)).

Objectives

- Help you keep your application easy to develop, understand, and maintain
- Structure of a Clean Architecture project
- Use of CQRS (Command Query Responsibility Segregation)

- Implementation of unit and integration tests

Target audience

- Web and application developers
- Architects

Prerequisites

- Knowledge of object-oriented programming
- Knowledge of the .NET environment
- [Test My Knowledge](#)

Technical prerequisites

- Visual Studio 2022 (Community minimum)
- Docker Desktop to be able to launch images (optional)

Our Clean Architecture with .NET Core Training Program

[Day 1 - Morning]

Introduction to architecture: How does architecture impact your business?

- Definition of software architecture and its strategic role in a project
- Direct impact on maintainability, quality, and speed of delivery
- Hidden costs of poor architecture (technical debt, instability, costly redesigns)
- Relationship between architecture, agility, and the ability to evolve the product
- Interactive mini-exchange: roundtable discussion on the difficulties encountered by participants in their projects

Analysis of legacy code repositories in MVC and N-Tier

- Review of the principles of MVC and N-Tier architectures
- Identification of common pain points:
 - Strong coupling between layers
 - Multiplication of business logic in controllers or in the database
 - Difficulties with unit testing and evolution

- Critical reading of excerpts from legacy code in .NET (MVC/N-Tier)
- Highlighting symptoms of advanced technical debt (duplicate code, circular dependencies, increasing complexity)
- Comparison with the objectives of a modern architecture (independence, testability, scalability)
- Hands-on workshop: group exploration of a simplified legacy repository and discussion of issues

[Day 1 - Afternoon]

Definition of the "common thread" project and start of coding

- Presentation of the common thread project that will serve as a basis throughout the training
- Clarification of functional requirements and initial business rules
- Implementation of the .NET solution (project skeletons, basic dependencies)
- Hands-on workshop: collective creation of the project skeleton and initial business entities.

Basic principles of automated testing

- Why test? Difference between manual and automated testing
- Types of testing: unit, integration, end-to-end (quick overview)
- Benefits in terms of quality, maintainability, and confidence in the code
- Introduction to xUnit in the .NET ecosystem
- Rules for writing effective tests: AAA (Arrange, Act, Assert), readability, independence
- Hands-on workshop: writing the first unit tests on the core project code, using the TDD method.

[Day 2 - Morning]

Presentation of SOLID principles

- Why SOLID remains an essential foundation for modern architectures
- Details of each principle with simple examples in C#
- Typical anti-patterns when SOLID is not followed
- How to detect when a principle is not being followed
- Practical exercise: identifying SOLID violations in the repositories seen the day before

Presentation of the fundamental concepts of "modern" architectures (Ports & Adapters, Clean Architecture, Hexagonal Architecture)

- Observation of the limitations of the MVC/N-Tier approaches seen the day before
- Key principles: separation of concerns, inversion of dependencies, domain independence
- Ports & Adapters (Hexagonal Architecture):
- Ports = business interfaces
- Adapters = technical implementations (DB, API, UI)

- Clean Architecture: organization into concentric layers (Domain, Application, Infrastructure, UI)
- Hands-on workshop: schematization of concepts on the common thread project

[Day 2 - Afternoon]

Implementation of a "Ports & Adapters" architecture for the core project

- Creation of contracts (ports) to isolate the domain
- Implementation of the first adapters (in-memory, simulated API)
- Integration of existing code (common thread) into this new structure
- Immediate benefits: testability, implementation substitution
- Hands-on workshop: implementation of a use case with Ports & Adapters in the Fil Rouge project

Evolution of the architecture towards a "Clean Architecture"

- Project structure in concentric layers: Domain / Application / Infrastructure / UI
- Dependency rules: the domain does not depend on anything
- Gradual adaptation of existing code to comply with Clean Architecture
- Comparison with Ports & Adapters: what changes and what stays the same
- Hands-on workshop: migrating the main project to a complete Clean Architecture

[Day 3 - Morning]

Introduction to CQRS (Command Query Responsibility Segregation)

- Reminder of the principle of separation between reading and writing
- Differences between the classic CRUD approach and CQRS
- Benefits: simplified use cases, improved scalability, alignment with Clean Architecture
- Implementation of handlers (CommandHandler, QueryHandler) in a .NET project
- Hands-on workshop: implementation of a use case from the main project with CQRS

Advanced CQRS and best practices

- Management of specific DTOs for Command and Query
- Impact on testing: isolation and clarity
- When CQRS is relevant... and when it isn't
- Hands-on workshop: adding a Query and extending the core project with dedicated unit tests

[Day 2 - Afternoon]

Introduction to BDD (Behavior Driven Development)

- Differences between TDD and BDD: focus on business language and collaboration
- Role of BDD in validating business rules
- Presentation of Gherkin: structured natural language (Given/When/Then)
- Writing scenarios that are readable by business users and executable by developers
- Support tools in the .NET ecosystem (SpecFlow, ReqNRoll, etc.)
- Hands-on workshop: collaborative writing of Gherkin scenarios for a use case from the main project and automation of acceptance tests with Gherkin

Practical implementation of BDD with the core project

- Transformation of scenarios into executable automated tests
- Demonstration: a Gherkin scenario that directly drives .NET code
- Review of the benefits: business communication, living documentation, increased quality
- Limitations and pitfalls to avoid (overly technical scenarios, duplication with unit tests)
- Practical workshop: implementation of an end-to-end Gherkin scenario on the core project

Companies involved

This training is aimed at both individuals and companies, large or small, wishing to train their teams in new advanced IT technology or to acquire specific professional knowledge or modern methods.

Placement at the start of training

The placement test at the start of the training course complies with Qualiopi quality criteria. Once they have finalized their registration, learners receive a self-assessment questionnaire that allows us to assess their estimated level of proficiency in different types of technologies, as well as their expectations and personal objectives for the upcoming training course, within the limits imposed by the selected format. This questionnaire also allows us to anticipate certain connection or internal security issues within the company (intra-company or virtual classroom) that could be problematic for the monitoring and smooth running of the training session.

Teaching methods

Practical training: 60% practical, 40% theory. Training materials distributed in digital format to all participants.

Organization

The course alternates between theoretical input from the trainer, supported by examples and reflection sessions, and group work.

Validation

At the end of the session, a multiple-choice questionnaire is used to verify that the skills have been correctly acquired.

Certification

A certificate will be issued to each trainee who has completed the entire training course.