Clean Architecture

CS 346 Application Development

https://student.cs.uwaterloo.ca/~cs346

Contents

Introduction

What is it?

Clean architecture is an architectural style that is commonly used with modern mobile and desktop applications.

It was introduced by Robert C. Martin in a 2012 blog post [1], and refined over subsequent years [2].

Clean architecture

- Generalized version of a Layered Architecture.
- Can handle multiple generalized services.
- Designed to solve cohesion and coupling issues.

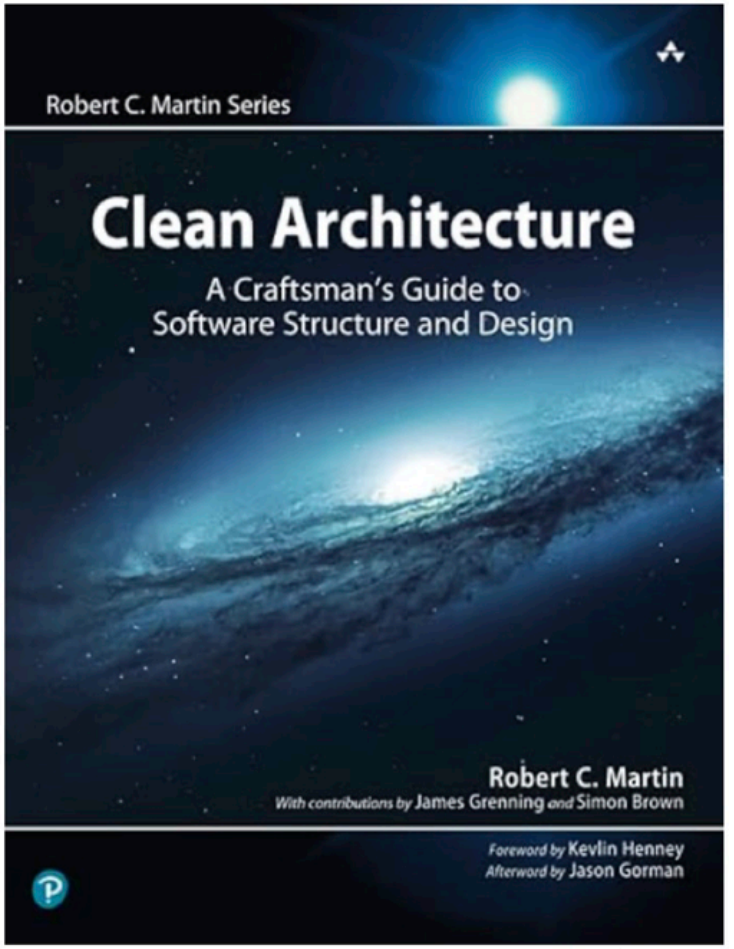


Figure 1: Clean Architecture, 2017 [2]

Benefits

Clean Architecture aims to improve on a layered architecture in few ways:

- **Independence from frameworks**. The architecture does not depend on a particular set of libraries for its functionality.
- **Improved Testability**. The business rules can be tested without the UI, database, web server, or any other external element.
- **Independence from the UI**. The UI can change easily, without changing the rest of the system. A web UI could be replaced with a console UI, for example, without changing the business rules.
- **Independence from the database**. You can swap out Oracle or SQL Server for Mongo, BigTable, CouchDB, or something else. Your business rules are not bound to the database.

Clean Architecture

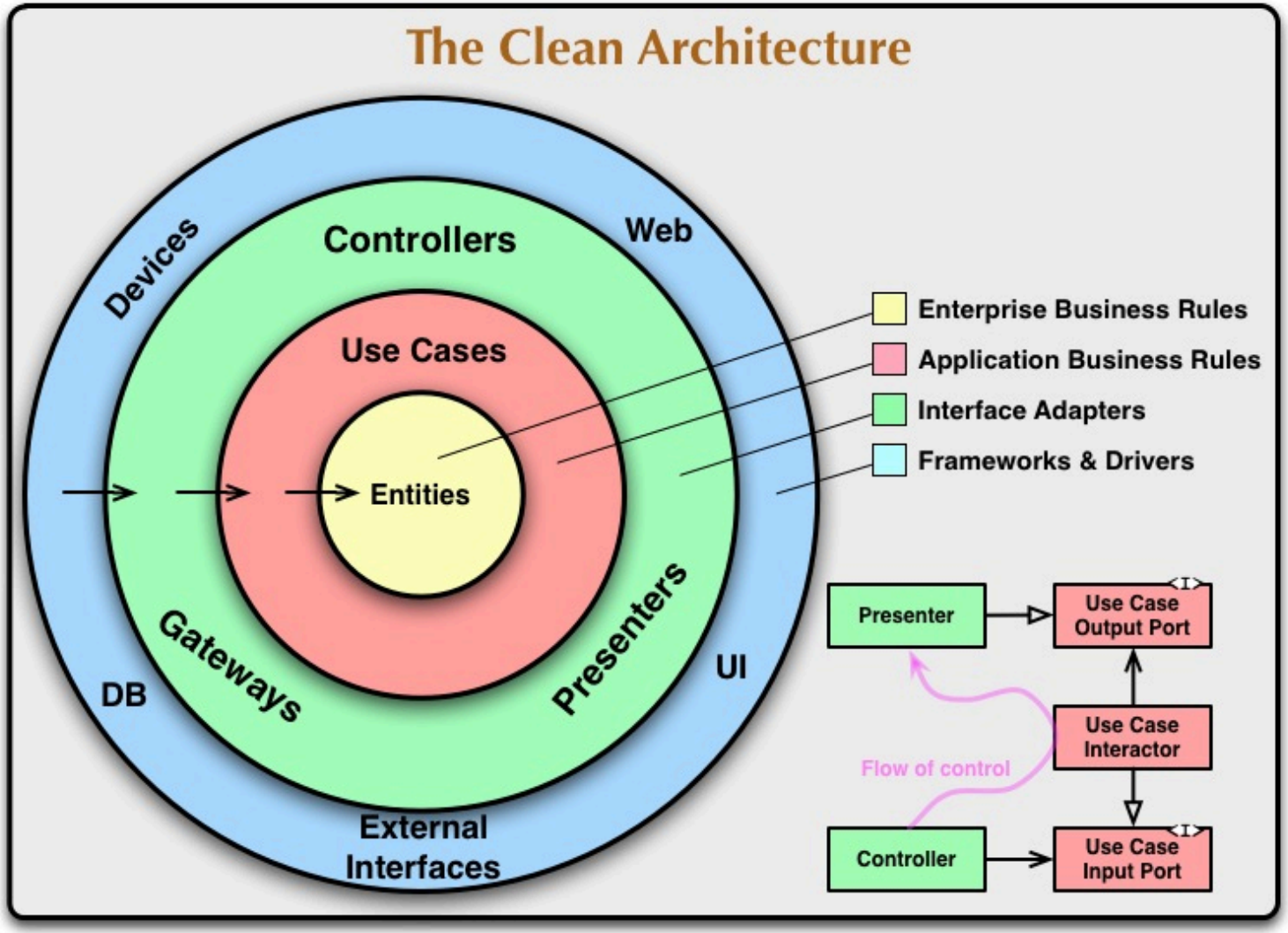


Figure 2: Courtesy of the Clean Coder Blog [1]

Layers

We have multiple layers in every application:

Entities (Core)

- Business classes that other layers use. These are often data classes.

Use Cases (Inside)

- Core application logic that facilitates data flow to and from the entities.

Controllers (Middle)

- Adapters to map data from entities to a format for external services.

Interfaces (Outside)

- Interfaces to external services e.g., databases, web APIs.

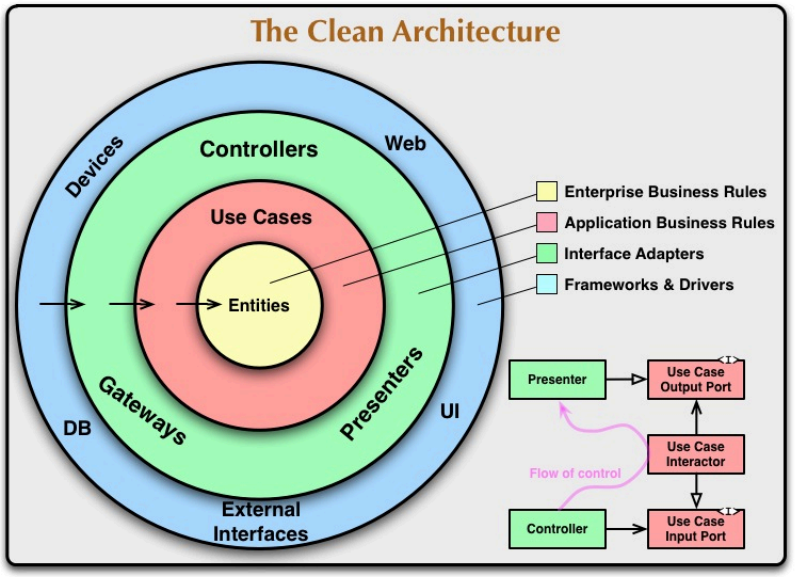


Figure 3: This is similar to a layered architecture, except that we push all external dependencies to the outer layers.

Dependencies

The key to this architecture is careful control of dependencies.

The **dependency rule** describes the relationship between layers:

- Dependencies only flow inwards.
- i.e., classes can only `import` from the same layer or a more inner layer.

What are the consequences of this?

- All frameworks and services become loosely coupled.
- e.g., database, web service, user interface.

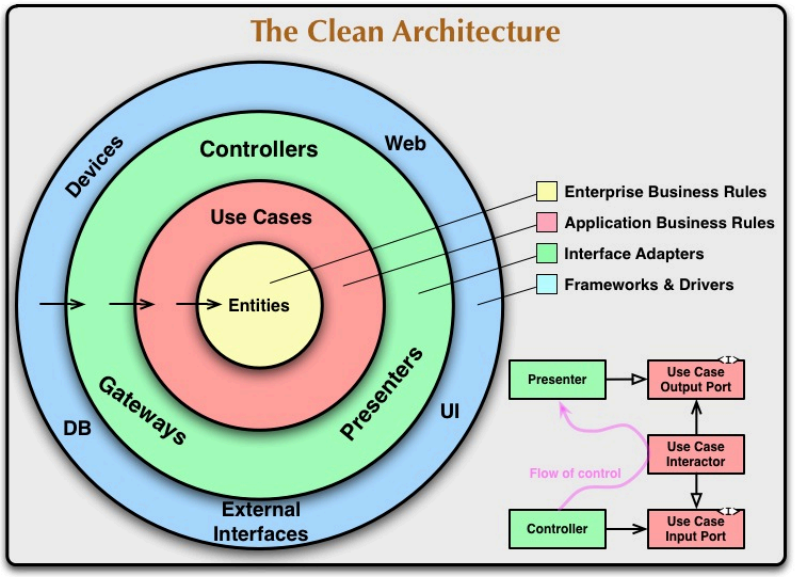


Figure 4: Outer layers can directly access inner layers, but not the other way around.

Dependency Inversion

To make loose coupling work, we need a lightweight communication mechanism.

We'll use dependency inversion as much as we can.

- Every major class in the outer layers has an interface.
- Innermost layers can use that interface to describe a service or framework that they support.
- We define our outermost classes as realizations of those interfaces.

DI means that we can pass around references to service interfaces.

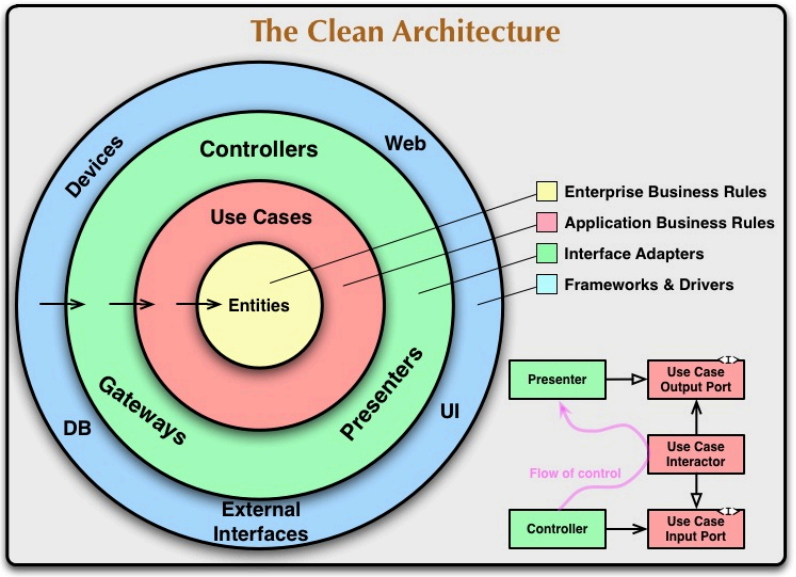
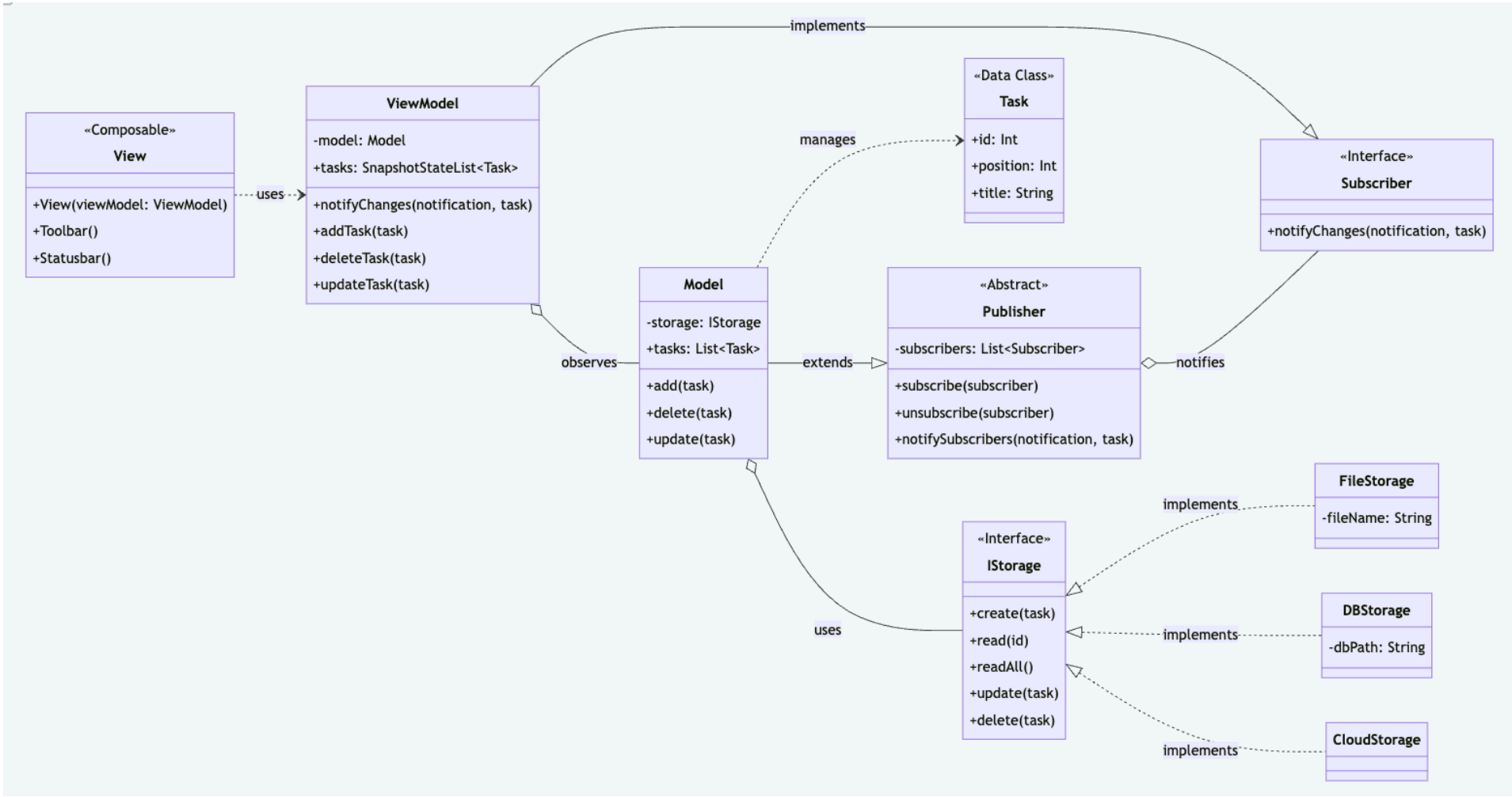


Figure 5: Every major class has an interface that describes its behaviour. We use DI to pass around object references.

Example

`taskit`



Bibliography

[1] R. C. Martin, “The Clean Architecture.” [Online]. Available: https://blog.cleancoder.com/uncle-bob/2012/08/13/the-clean-architecture.html

[2] R. C. Martin, *Clean Architecture - A Craftsman's Guide to Software Structure and Design*. Pearson, 2017.