App Architecture

Created by - Mohammad Mugish

GOALS

1. **Difference between Architecture and Design Pattern**
2. **What is Clean Architecture and Why we are planning to use this**
3. **How Clean Architecture Works in Mobile Application with MVVM**

1. Difference between Architecture and Design Pattern

“The word “architecture” is often used in the context of something at a high level that is divorced from the lower-level details, whereas “design” more often seems to imply structures and decisions at a lower level. But this usage is nonsensical when you look at what a real architect does.”

Excerpt From
Clean Architecture
Robert C. Martin

Architecture

1. Architecture is the overall structure of software
2. Architecture is the structure of the software system in its entirety.
3. The developer chooses different design patterns according to the architecture specifications and requirements.

Design Pattern

1. Design patterns are concerned with how the components are built.
2. It’s about a particular solution.

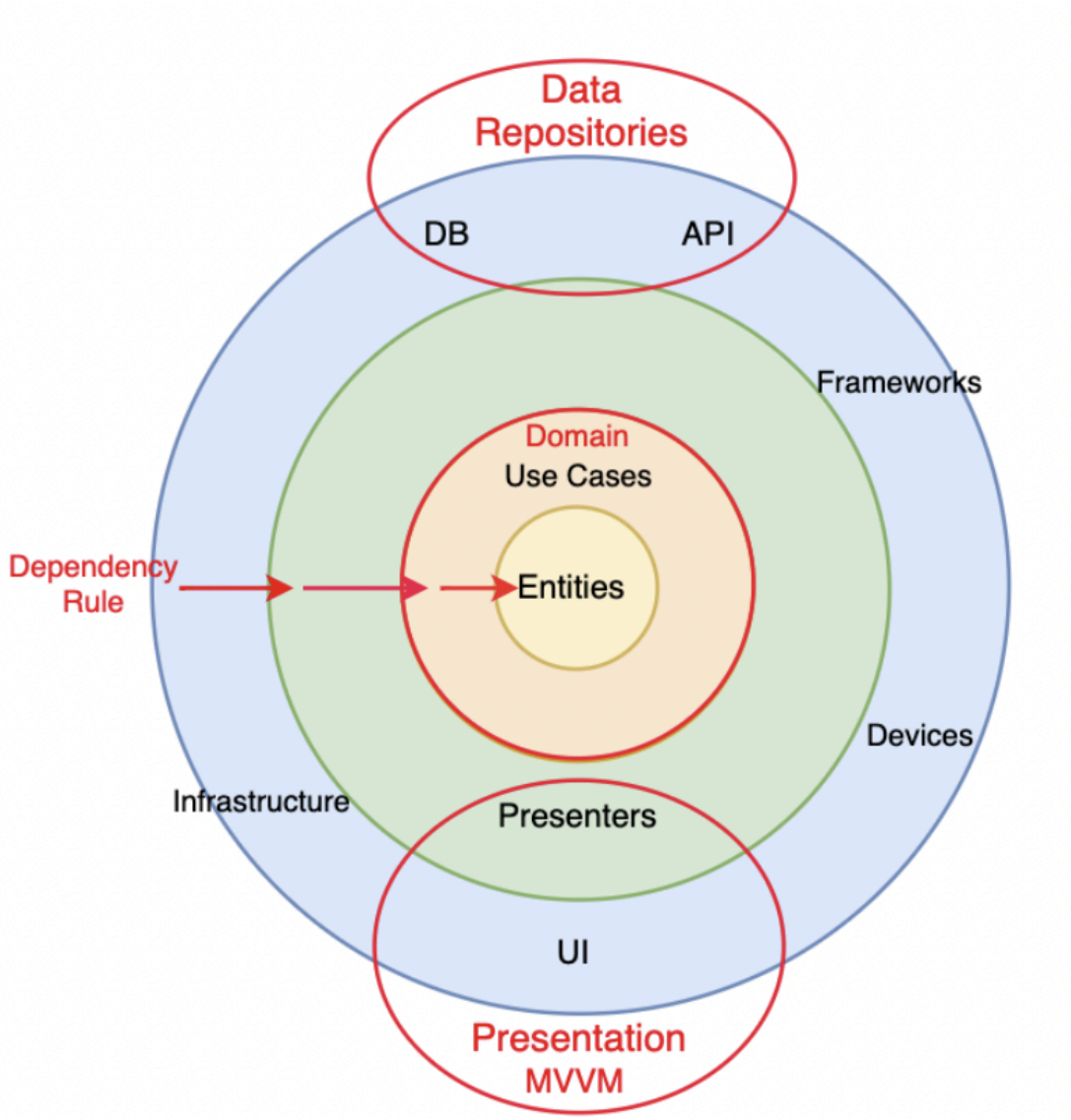
Difference

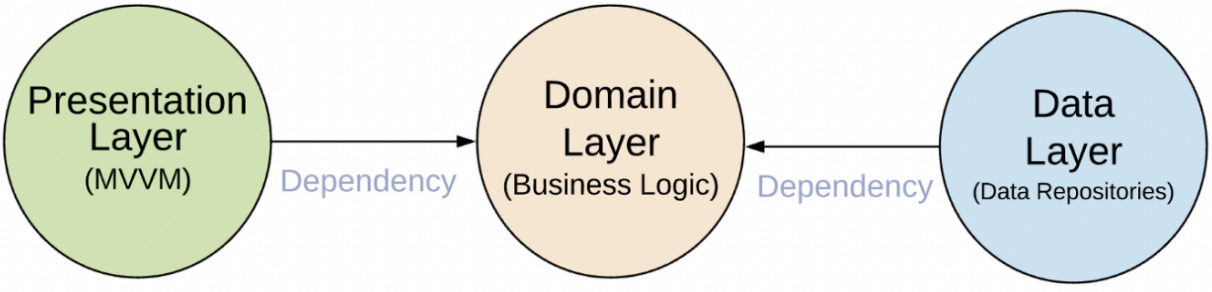
- Architecture comes in the Designing phase and Design Patterns comes in the Building phase.
- Architectural pattern is like a blueprint and design pattern is actual implementation.
- Architecture is the base to which everything else adheres and design pattern is a way to structure classes to solve common problems.
- All **Architecture is design pattern** but all **design patterns can not be architecture.** Like MVC can come under both. However, a singleton design pattern can not be an architectural pattern.

2. What is Clean Architecture and Why we are planning to use this

This Architectures produce systems that are:

- ➔ **Independent of Frameworks.** The architecture does not depend on the existence of some library of feature laden software. This allows you to use such frameworks as tools, rather than having to cram your system into their limited constraints.
- ➔ **Testable.** The business rules can be tested without the UI, Database, Web Server, or any other external element.
- ➔ **Independent of 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.
- ➔ **Independent of 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.
- ➔ **Independent of any external agency.** In fact your business rules simply don’t know anything at all about the outside world.


Data
Repositories
DB
API
Frameworks
Domain
Use Cases
Entities
Dependency
Rule
Devices
Infrastructure
Presenters
UI
Presentation
MVVM


Presentation
Layer
(MVVM)
Dependency
Domain
Layer
(Business Logic)
Dependency
Data
Layer
(Data Repositories)

A. Entities

An entity can be an object with methods or a set of data structures and functions. They encapsulate the most general and high-level rules. They are the least likely to change when something external changes.

B. Use Cases

The software in the use cases layer contains application-specific business rules. It encapsulates and implements all of the use cases of the system.

C. Interface Adapters

The software in the interface adapters layer is a set of adapters that convert data from the format most convenient for the use cases and entities to the format most convenient for some external agency such as the database or the web.

D. Frameworks

The framework layer is where all the details go. The web is a detail. The database is a detail. We keep these things on the outside where they can do little harm. Generally, you don’t write much code in this layer, other than glue code communicating to the next circle inward.