Clean Architecture for Android

Creating scalable, maintainable and testable native Android apps

2nd Edition

Eran Boudjnah

www.bpbonline.com

Second Revised and Updated Edition 2026

First Edition 2023

ISBN: 978-93-65891-676

LIMITS OF LIABILITY AND DISCLAIMER OF WARRANTY

The information contained in this book is true and correct to the best of author's and publisher's knowledge. The author has made every effort to ensure the accuracy of these publications, but the publisher cannot be held responsible for any loss or damage arising from any information in this book.

All trademarks referred to in the book are acknowledged as properties of their respective owners but BPB Publications cannot guarantee the accuracy of this information.

Dedicated to

*My endlessly supportive wife, **Lea***

Forewords

- There is a point in every Android developer's journey when the code starts to feel heavier than it should. The feature is simple, but the implementation drags. A small change in UI breaks something in the data. Business rules are scattered. No one is sure what touches what anymore.

  At some point, the joy of building turns into the stress of maintaining.

  That is usually when people start searching for better architecture. Some end up discovering Clean Architecture through blog posts or conference talks. Others try to piece it together through trial, error, and code reviews that go in circles. And many never get the time or clarity to understand what clean even means in the context of real-world Android apps.

  This book is a gift to all of them.

  It does not assume you already know Clean Architecture, but it also does not waste your time if you do. Instead, it offers something rare: a complete, opinionated, and practical walkthrough of Clean Architecture as it applies to modern Android development. It embraces tools we actually use—Hilt, Compose, Kotlin DSL—and gently reshapes Uncle Bob's core ideas to fit the messy, asynchronous, Google-evolving reality of our day-to-day work.

  The author is not preaching from a pedestal. He is writing from the trenches. With over two decades of experience behind the keyboard, he has seen what bloats, what breaks, and what quietly survives release after release. His approach is shaped not just by principles, but by actual pain, and the relief of finding what works.

  This is not a silver bullet. It is better than that.

  It is a steady path out of chaos. It is a set of small, conscious decisions that add up to big clarity. It is architecture that lets you move fast—and keep moving.

  Whether you are a developer trying to clean up your first app or a team lead looking for a sane structure to scale with, this book will not just show you how. It will help you think differently about what good code actually means.

  Read it slowly. Rethink a few things, and maybe next time the code starts to feel heavy, it will not be because it is broken, but because it is solid.

  - Gaurav Thakkar
  Mobile applications specialist at BookMyShow

- In software engineering, we constantly navigate the tension between immediate delivery and long-term viability. The initial velocity of a project is often seductive, but without a solid architectural design, it inevitably gives way to the corrosive effects of software entropy. As the quote from Norman Foster in the introduction to this book reminds us, we must design for a future that is essentially unknown. This principle is the very soul of professional software development.

  The journey of our industry in Android development has been a collective search for this architectural ideal. We evolved from monolithic UI components, where logic and display were dangerously intertwined, toward more disciplined approaches. We learned to separate concerns, first by creating orchestrating classes to direct our views, and later by adopting reactive models to create a seamless flow of data to the UI.

  This is the critical problem that this second edition of Clean Architecture for Android solves with such clarity and conviction. It moves beyond transient patterns to advocate for a set of enduring principles. At its heart is the Dependency Rule, a non-negotiable mandate that all dependencies must point inward, toward the center of the application. This book provides the definitive blueprint for building a system around an inviolate Domain layer, where your pure business logic resides, entirely insulated from the outside world.

  You will learn to treat the UI, the database, and the network as what they are: implementation details. The Data layer becomes a sophisticated adapter, negotiating with external systems and translating their chaos into the clean, stable language of your Domain. The Presentation layer becomes a thin servant to this core logic. By enforcing this separation, you build an application that is not just testable and maintainable, but truly resilient and adaptable.

  This book is more than a technical manual; it is a guide to professional practice. It offers the disciplined framework required to build sophisticated, large-scale applications that are designed to last. For any developer who considers their work a craft, the principles within these pages are not just recommended; they are essential.

  - Jesus Rodriguez
  Senior Android Developer Manager

About the Author

Eran Boudjnah has been developing apps and leading mobile teams for a wide range of clients, from start-ups (JustEat, Plume Design) to large-scale companies (Sky, HSBC) and conglomerates since 1997. He has been working with Android since around 2013.

Eran is a developer with almost three decades of experience in developing mobile applications, websites, desktop applications, and interactive attractions. He is passionate about board games (with a modest collection of a few hundred games) and has a 90's Transformers collection on display, of which he's quite proud.

Eran lives in Brentwood, Essex, in the United Kingdom with Lea, his incredibly supportive wife.

Acknowledgement

On a personal note, writing a book about Clean Architecture proved to be quite an exciting journey. It was a great opportunity to challenge ideas and principles. None of it would have been possible if not for many people who challenged me, learned with me, helped me grow, and supported me along the way. While I cannot name all of them, I would like to name a few. If I have not mentioned you, I do apologize. I got help from so many people.

In no particular order, my gratitude goes to Jose Antonio Corbacho, Davide Cirillo, Sébastien Rouif, Amr Yousef, Tim Hepner, Manroop Singh, Muhamed Avdić, and Mahmoud Al-Kammar.

I am thankful to Igor Wojda for reviewing the first edition of this book so thoroughly. It would not have been as accurate or detailed if not for his invaluable feedback.

I would also like to thank my wife, Lea, who supported me throughout this process, which was quite demanding at times.

Lastly, thank you for taking the time to read this book. It is my hope that, having read it, you are now comfortable with Clean Architecture in the Android world. Maybe with your own ideas and experience, we can keep evolving it and make it ever better.

Preface

I asked him if he'd come to clean the windows, and he said no, he'd come to demolish the house. He didn't tell me straight away, of course. Oh no. First, he wiped a couple of windows and charged me a fiver. Then he told me.

- Douglas Adams

Clean Architecture is not new. It has been around since 2012. Applying it to Android is not a new idea either, and has been done by many teams over the years.

What is surprising is this: despite Clean Architecture being around for so long, there is still no definitive source if you want to figure out how to implement it in your project. At the time of writing, if you searched for one, you would find blog posts, articles, online courses, and a couple of books, published close to when the first edition of this book was published. Each one of those sources would suggest a somewhat different approach. All are valid, and I have seen them all implemented with varying degrees of success. None answered all the questions that I had when I got to the actual development of real-life projects.

This is a real shame because I could have used such a source. When a client of mine wanted to merge two of their Android apps into one, it sure would have helped us. Unfortunately, it was not there, and the merging process ended up dragging on for years. In hindsight, had the projects adopted Clean Architecture, the process would have been much smoother. We could have migrated it in parts, feature by feature, layer by layer, and plugged the common code in. I cannot begin to measure how much time and money could have been saved.

However, there was no definitive source of information for Clean Architecture back then, and we could not answer the questions that we had in a satisfactory way. We could not get a clear picture of how that solution would work for us.

We had many questions: what exactly is business logic? Where do I draw the line between business logic and presentation logic? Just how much responsibility should the UI have? How do models travel between the layers? How much logic should I have in my Data layer? What do I do if the backend work is still in progress? Having this book would have answered all our questions.

Another client of mine decided to go forward with Clean Architecture despite not having a clear understanding of the pattern. I cannot blame them for the reason mentioned previously. They made a good effort. Unfortunately, they also made quite a few mistakes. Unfortunately, this meant that the code was gradually becoming harder to maintain. Those mistakes alone were enough to lead them down a path of a full code rewrite. In retrospect, they are happy

with their decision to rewrite the project. They are moving much faster now and scaling rapidly without having to slow down to onboard new developers. The architecture is self-explanatory, and all you need to implement a new feature is to look at one of the many other examples in the code. The test coverage and testing policy provide them with a high level of confidence.

As an Android consultant, I was fortunate enough to work with many skilled developers from all over the world over the course of over 12 years. Out of the 12 years, I spent six implementing Clean Architecture. Together with my colleagues, we have iterated, reiterated, polished, and rewritten our implementation. We had to answer all the questions mentioned above and many others.

It took four years for me to feel ready to share my understanding of architecture with you. It took another two to complete this second edition. I hope reading about my experience will help new developers as well as veterans. I aim for this book to be a resource you could come back to whenever you are not sure about any part of your architecture.

This book is divided into 15 chapters. We will cover the Clean Architecture principles as applied to Android and look at an end-to-end implementation. We will continue to explore testing as well as failures and exceptions. Then, we will demonstrate how to implement a new feature. Finally, we will discuss migrating existing projects and anything important that would not fit in any of the earlier sections.

Chapter 1: Introduction- Covers the motivation for writing the book, its key benefits for the reader, and sets the expectations in terms of how the book is going to be structured. It provides some background on Clean Architecture and Clean Code before we dive into greater detail. The first section will provide three real-life examples of where Clean Architecture could have saved (unnamed) clients from having to rewrite their app due to requirement changes.

Chapter 2: Clean Architecture Principles- This chapter will break down Clean Architecture into its individual layers as they are applied in the Android world. We will cover the responsibility of each layer and the components that live in each one.

Chapter 3: The Domain Layer- It is the first of five chapters in which we break down the key components of each Clean Architecture layer and how they all come together in a working app. In this chapter, we will discuss the Domain layer and cover usecases, repository interfaces, and domain models.

Chapter 4: The Presentation Layer- In this chapter, we continue our review of the Clean Architecture components. We cover viewmodels, presentation models, and bidirectional mapping between domain and presentation models.

Chapter 5: The UI Layer- This chapter follows up from the previous chapter and moves on to explain the UI layer. In it, we go over Activities, Fragments, Views, and composables, as well as UI models and bidirectional UI to presentation mappers.

Chapter 6: The DataSource and Data Layers- After exploring the architecture from the domain all the way to the UI, in this chapter, we go the other way and explore the data side of the app. We learn about repository implementations, datasources, and finally API and database dependencies. We will encounter the models that go in the datasource and data models, and the associated mapping for these models.

Chapter 7: Dependency Injection and Navigation- In this fifth and last chapter breaking down the Clean Architecture implementation, we cover the different options that we have for implementing navigation and dependency injection, bringing the different parts of the app together. We will also demonstrate how navigation can be done in this architecture.

Chapter 8: Unit Testing- We will demonstrate how Clean Architecture makes unit testing easier. We will start by briefly discussing the value of writing tests. We will then go layer by layer, discuss the components that can be tested, what needs to be covered by tests, and provide examples of how those tests look, using Junit 4 and Mockito.

Chapter 9: End-to-end Testing- We will discuss the parts that are harder to test: the UI and the integration of all the different parts. We will explain the robot pattern and provide examples for testing composables.

Chapter 10: Mocking the Server- We will continue covering integration tests and We will show the basics of mocking web server responses using MockWebServer.

Chapter 11: Failures and Exceptions- This chapter will cover the difference between failures and exceptions and how both can be handled in Clean Architecture. We will provide an example of an API timeout exception to demonstrate exceptions and a user not found error to demonstrate failures.

Chapter 12: Implementing a New Feature- We will demonstrate the implementation of a new feature in an existing app. We will focus on the best order in which to go about the task and explain why that order is important following an approach that was proven to work well for a single developer as well as large teams of developers.

Chapter 13: Dealing with Changes- This chapter is all about changing requirements. In it, we will see how Clean Architecture rewards us for our efforts by making changes easier. This chapter will provide two concrete examples: replacing a datasource and updating the user interface.

Chapter 14: Migrating an Existing Project- In this chapter, we will briefly touch on the common architectures out there. We will then discuss how a gradual migration can be performed. This

will allow the reader to switch to Clean Architecture without it being a colossal endeavour. We will provide examples of migrating from MVVM and MVP to Clean Architecture (with MVVM or MVP) by introducing usecases for new requirements or while working on bug fixes. We will also show how logic can be moved out of poorly written usecases and into the Data layer. We will emphasise the importance of tests being in place to protect us from breaking the existing behaviour.

Chapter 15: Other Bits and Bobs- This is the final chapter. In it, we will mention that Clean Architecture is a tool. It is there to serve us and help us structure problems, not to tie our hands. We will remind ourselves that the other tools that we have acquired along our journey as developers are all still valuable and can still be applied. The SOLID principles, DRY and KISS are all still valid and should be considered when implementing Clean Architecture.

While this book is by no means gospel, I hope that you will find reading it helps all the pieces fall into place in your head. Ideally, when you later face a new feature request, you would easily visualize its implementation details in your mind.

Convention

This book follows a few conventions that are worth mentioning.

Many of the code snippets are incomplete. They highlight parts of the file that are worth discussing. The complete file as well as the whole project in the context of individual chapters can be found in the GitHub repository.

References to code are `highlighted`.

Important terms and library names are bold.

Emphasized words are in italics.

Tips and key takeaways are highlighted like this sentence.

The implementation of Clean Architecture presented in this book is an opinionated one. This means that in places it may be stricter than the official Clean Architecture model. This choice is based on accumulated experience from many past projects that I have been involved with over the years, in which different approaches were explored. You are welcome to stray off the path laid out in this book, but you will be doing so at your own peril.

Finally, I had to make some technical choices for this book:

- For dependency injection, I chose to use Hilt. This does not mean that you have to use Hilt, too.
- As an architectural pattern, I will mostly focus on **Model-View-ViewModel** (**MVVM**).
- For Gradle scripts, I will I will be using the Kotlin **Domain Specific Language** (**DSL**) rather than the Groovy one. It was introduced in version 3.0 of the Gradle Build Tool as far back as August 2016 and has been the default for new builds since 2023.

Code Bundle and Coloured Images

Please follow the link to download the
Code Bundle and the ***Coloured Images*** of the book:

https://rebrand.ly/49cc1f

The code bundle for the book is also hosted on GitHub at
https://github.com/bpbpublications/Clean-Architecture-for-Android-2nd-Edition
In case there's an update to the code, it will be updated on the existing GitHub repository.

We have code bundles from our rich catalogue of books and videos available at **https://github.com/bpbpublications**. Check them out!

Errata

We take immense pride in our work at BPB Publications and follow best practices to ensure the accuracy of our content to provide an indulging reading experience to our subscribers. Our readers are our mirrors, and we use their inputs to reflect and improve upon human errors, if any, that may have occurred during the publishing processes involved. To let us maintain the quality and help us reach out to any readers who might be having difficulties due to any unforeseen errors, please write to us at: **errata@bpbonline.com**

Your support, suggestions and feedback are highly appreciated by the BPB Publications' Family.

Piracy

If you come across any illegal copies of our works in any form on the internet, we would be grateful if you would provide us with the location address or website name. Please contact us at **business@bpbonline.com** with a link to the material.

If you are interested in becoming an author

If there is a topic that you have expertise in, and you are interested in either writing or contributing to a book, please visit **www.bpbonline.com**. We have worked with thousands of developers and tech professionals, just like you, to help them share their insights with the global tech community. You can make a general application, apply for a specific hot topic that we are recruiting an author for, or submit your own idea.

Reviews

Please leave a review. Once you have read and used this book, why not leave a review on the site that you purchased it from? Potential readers can then see and use your unbiased opinion to make purchase decisions. We at BPB can understand what you think about our products, and our authors can see your feedback on their book. Thank you!

For more information about BPB, please visit **www.bpbonline.com**.

Join our Discord space

Join our Discord workspace for latest updates, offers, tech happenings around the world, new releases, and sessions with the authors:

https://discord.bpbonline.com

Table of Contents

Chapter 1
Introduction

As an architect, you design for the present, with an awareness of the past for a future which is essentially unknown.

- Norman Foster

Introduction

Before we dive into concrete examples and code, we should have a bit of background. In this chapter, we will learn about the history of the Android operating system, have an overview of Clean Architecture, and compare it to **Model-View-ViewModel** (**MVVM**). Finally, we will discuss migrating existing projects to Clean Architecture and touch on the importance of Clean Code.

Structure

In this chapter, we will cover the following topics:

- History of Android
- Clean Architecture overview
- Clean Architecture vs. MVVM
- So now I must rewrite my project
- Clean Code

Objectives

By the end of this chapter, readers will have a general idea of what Clean Architecture is. Readers should also have a rough idea about how Clean Architecture could be introduced into existing projects. Lastly, I will share with you my view on code quality and how it affects the final product.

History of Android

Android is quite a mature platform and was unveiled in November 2007. The first device running on Android was the HTC Dream, shown in *Figure 1.1*, which was launched in September 2008. Ever since Android came out, there have been Android developers. In the early days, no architecture dominated the Android market. It was quite often that you would find massive **`Activity`** *god classes* holding the entire logic of the app.

***Figure 1.1**: The HTC Dream*

It did not take long until we all started realizing that this would not work. As soon as apps had any complexity to them, it became impossible to maintain or scale the code. Writing tests was a nightmare.

The first architecture to take the Android world by storm was **Model-View-Presenter** (**MVP**), illustrated in *Figure 1.2*. It gave us some structure. Code did not have to live inside the **`Activity`** anymore. We started having components with clear responsibilities. Our business logic would go in the **presenter**, along with the presentation logic. We could unit-test the presenter. Some implementations moved the business logic to the **model**. The **`Activity`**, **`Fragment`**, and custom **`View`** classes (Android **`View`**, not MVP View) would implement MVP-**view** interfaces, which the presenter used to drive our Android UI classes. The model used Retrofit services and database interfaces to access data. Only now, the presenter was getting bloated, and so were the view interfaces.

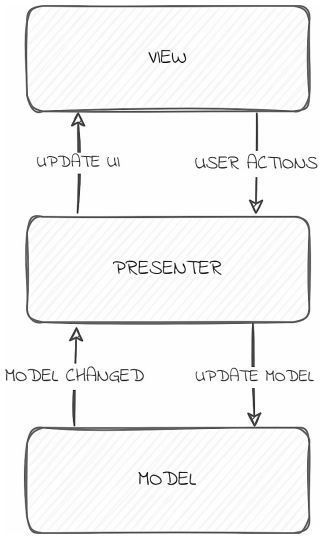

Figure 1.2: *Model-View-Presenter*

Along came Google's Architecture components, making our lives a bit easier. The great migration from MVP to **Model-View-ViewModel** (**MVVM**), which is illustrated in *Figure 1.3*, began. Around mid-2017, if you were brave enough, you could have picked up the library, still in Alpha at the time. By November that year, you could have started using the first stable release. We got the lifecycle-aware **`ViewModel`** class (with its own host of issues). **LiveData** made observing the **viewmodel** reasonably straightforward. Unit-testing the viewmodel was also easy enough. However, viewmodels did not solve *all* our problems. They could still get incredibly bloated, carry way too much responsibility, and hold too much code.

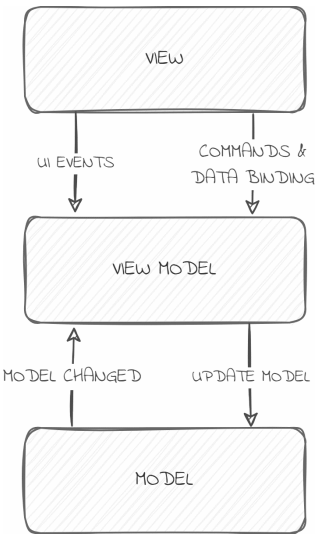

Figure 1.3*: Model-View-ViewModel*

It was not all bad. Not all architectures suffered from bloated presenters and viewmodels, some started adopting **Clean Architecture** to varying degrees. Some projects had **repositories** abstracting the different **datasources** for the presenters and viewmodels. Some even had **usecases**, connecting presenters and viewmodels to repositories.

This is not all that surprising, because Clean Architecture has been around since 2012, long before the great migration to MVVM, and only about three years after the earliest versions of Android were out.

Clean Architecture overview

Before going into what Clean Architecture is, let us first go over what a system architecture (the system being an application, in our case) is and why we need one.

An application, even the simplest one, has a few responsibilities. It needs to:

- Present the user with information.
- Collect input from the user, whether it is via taps, gestures, keyboard typing, or even voice commands.
- Process the input and perform actions based on that input.
- Quite frequently, it needs to send or retrieve data to or from a remote server or device.

Most applications have more responsibilities than that, and very few do not have all of them. Broadly speaking, it is possible to break down almost all applications into multiple responsibilities.

There are several reasons for us wanting to break down our application by responsibilities:

- **Testability**: We want to be able to test the different parts of the app in isolation.
- **UI independence**: We want to be able to change our UI without affecting the rest of the app.
- **Consistency**: When we or another developer approaches the code, having a consistent structure helps us understand the code and even predict it.
- **Maintainability**: This is tied to consistency. Consistent code is predictable and easier to work with. Generally, the fewer surprises you find in the code, the easier it is to maintain.

However, not all architectures are equal, and not all are fit for every purpose. This is crucial because it can be very hard to move away from many architectures once they are implemented.

This is where Clean Architecture shines. So, what is Clean Architecture, exactly?

On the surface, it is another system architecture. It is highly scalable and very easy to maintain. Since nothing comes for free, this comes at the cost of some initial writing overhead. However, that initial cost is easily paid back tenfold when you have maintainable and scalable code. It

is an established understanding that the ratio between reading and writing code leans heavily in favor of reading.

While implementing Clean Architecture, I learned something new about it. It is more than just architecture. It is a philosophy. Once adopted, it changes your way of thinking about applications and feature implementation. While I know that the saying is *if all you have is a hammer, everything looks like a nail*, that situation does not describe a problem where your tool is a Swiss army knife. Clean Architecture really does solve a lot of problems quite intuitively.

Enough theory, though. Let us take a look at how Clean Architecture looks. I have borrowed the following diagram from *Robert C. Martin* (Uncle Bob)'s blog[1]. I modified it slightly to better reflect our use of it when developing Android apps. The same is illustrated in *Figure 1.4*:

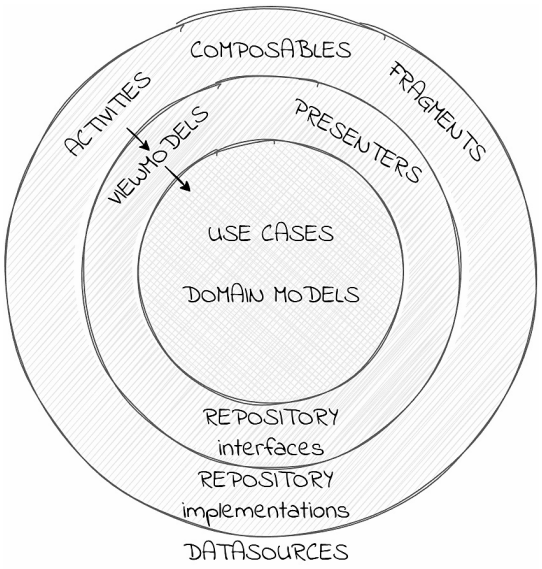

***Figure 1.4**: Clean Architecture (Android)*

On the outermost circle, we can see the representation of two layers: the UI layer and the Data layer. This circle includes Activity and Fragment classes and composable functions, as well as concrete repositories. The next circle, holding viewmodels and presenters, is the Presentation layer. The repositories are in this circle too, representing the Data layer. The innermost circle represents our Domain layer, where usecases and Domain models reside.

Note that the arrows are pointing inwards. They represent what Uncle Bob refers to as *The Dependency Rule*. You can read these arrows as *knows about*. So, for example, **Activity** classes know about **viewmodel** ones. **Viewmodel** classes, in turn, know about **usecase** ones. The opposite is not true: **viewmodel** classes do *not* know about **Activity** ones, and **usecase** classes do *not* know about **viewmodels** ones.

1 You can find the original blog post here: **https://blog.cleancoder.com/uncle-bob/2012/08/13/the-clean-architecture.html**

You may have noticed that the datasource classes are placed outside of the circles. This is because they represent external dependencies, and in that sense are exceptional. They follow a reversed Dependency Rule, as we will see shortly.

Think of the circles as protective layers. Changes are most often driven by the outer layers: the API we rely on changes. Our design requirements change. Even the Android **software development kit** (**SDK**) changes. The layer separation makes our usecases very stable and less likely to change as a result. The circles also represent levels of abstraction. The outside circle is highly technical, while the innermost one represents intentions and ideas, the business logic.

Another way of looking at Clean Architecture is to take a vertical slice, see *Figure 1.5*:

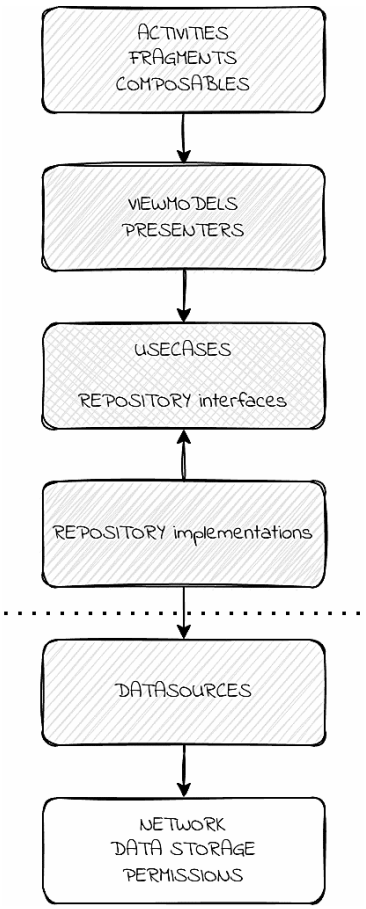

***Figure 1.5**: Clean Architecture vertical slice*

Looking at a vertical slice, we can see how the different components come together. This would be a good representation of a common feature. Remember, the arrows represent dependencies. *They do not represent how data flows*. Data can flow in both directions.

Note the dashed line between the repository implementations and the datasources. It is there to emphasize the exception to the Dependency Rule. Since datasources are not tied to particular features, it makes no sense for them to know about concrete repositories. It is also reasonable for us to think about them as external to a feature, and thus to the architecture circles. In actual code, we will see that concrete repository classes depend on datasource ones.

A quick word about layers: layers can take different forms. A layer can be a package in your project. It can be a Gradle module. In fact, it can also be multiple Gradle modules, each representing a part of a layer for a particular feature. In this book, we will have all layers per feature, and they will be contained in a Gradle module per layer per feature.

Let us look at an example. Let us say that we have a composable function with a button (see *Figure 1.6*). Let us also assume that we are using MVVM, and so we have a viewmodel class. When the user taps the button, the composable communicates that tap to the viewmodel. The viewmodel calls the execute function of a usecase object that describes that event. Let us say that the usecase class is called **`UpdateUserLastTapTimeUseCase`**. That usecase object then calls a function on the repository object. We can call it **`updateUserLastTapTime()`**. In turn, that function can trigger the storage of the current time in local storage or the cloud via a datasource object.

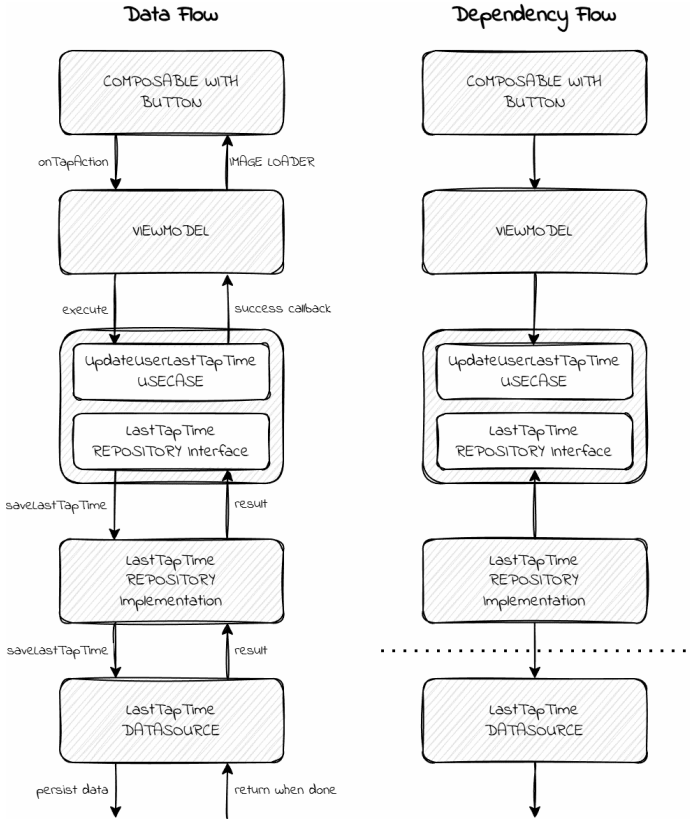

***Figure 1.6**: An example of Clean Architecture data and dependency flows*

The repository can then inform the usecase that it was successful (or that it failed), and the usecase can report that result to the viewmodel. The viewmodel would update its ViewState field, triggering an update to the composable.

However, wait, I hear you say. You said that the usecase does not know about the repository. How can the usecase make that call to the repository? To respect the Dependency Rule, we apply dependency inversion. Dependency inversion is one of the SOLID principles[2]. Instead of having usecases depend on concrete repository classes, we introduce repository interfaces. The repository interface will reside alongside the usecase class. Its concrete implementation, however, will live elsewhere. Now, the usecase class no longer relies on the repository implementation. This allows us to cross boundaries without violating the Dependency Rule.

This, in short, is what Clean Architecture is all about. Slice your app into layers, protect the business logic, and embrace change in the outer layers. In the following chapters, we will explore how this is done and what the benefits are.

Clean Architecture vs. MVVM

So, you already have an app developed. You followed Google's advice and have implemented MVVM. You have viewmodels. Your composable functions and **`Fragment`** and **`Activity`** classes are your views. Your viewmodels are probably quite large and communicate with repository objects directly. Maybe you even have usecases (or interactors, which are an alternative name to usecases). Does adopting Clean Architecture, as suggested in this book, mean that you have to say goodbye to MVVM? The short answer is no.

The long answer is that the two do not contradict. You can adopt Clean Architecture with MVVM as well as with MVP. The reason that these architectures can co-exist is that they overlap. An exception to this is Google's architecture, which contradicts Clean Architecture when it comes to the Dependency Rule and considers usecases to be optional.

If we break down MVVM into its components, we can see how the overlap works. The model moves to the Data layer. The viewmodel stays in what becomes the Presentation layer. The view moves to the UI layer. The only change we need to make is to introduce usecases between the viewmodel classes and the model if we do not have those yet. The usecases become our Domain layer. This is demonstrated in *Figure 1.7* (refer to *Figure 1.3* earlier in the chapter to see how the architecture looked before the change):

2 See here: **https://en.wikipedia.org/wiki/SOLID**