MVVM (Model View ViewModel) Architecture Pattern in Android

- It can be challenging for beginners to use MVVM.
- Testing for application can be difficult.
- Debugging issue due to XML.
- Very limited options are available in view methods.

Comparison with Other Architectures

a. MVP vs MVVM

MVP	MVVM
By employing Presenter as a route of communication between Model and View, it solves the issue of having a dependent View.	Because it leverages data binding, this architectural style is more event-driven and makes it simple to separate the view from the main business logic.
As a Presenter layer, observables are not required in this.	Because there is no Presenter layer in this, observables are required.
In MVP, UI is used more frequently.	In MVVM, there is no user interface.
The model layer sends the Presenter, who then sends it to the View, the response to the user's input.	The Model layer responds to the user's input by conducting operations and returning the result to the View.
More classes will be	More classes but less code per

included in the project file in addition to the code.	class will be found in the project file.

b. MVC vs MVVM

MVC	MVVM
MVC typically requires manual updating of the view by the controller.	MVVM utilizes data binding to establish a connection between the view and the ViewModel, allowing automatic synchronization.
In MVC, the view can contain some presentation logic.	In MVVM, the ViewModel handles most of the presentation logic, keeping the view more focused on displaying data.
Less testable in comparison to MVVM.	With MVVM, since the ViewModel contains most of the business logic, it can be easily unit tested without the need for the view or model.
MVC separates concerns by assigning specific roles to the model, view, and controller.	MVVM takes it a step further by introducing the ViewModel as a separate component responsible for presenting data to the view.
MVC, with its simpler structure, may be more suitable for smaller projects or applications with straightforward requirements.	MVVM can introduce additional complexity due to the introduction of the ViewModel layer.

c. MVI vs MVVM

MVI	MVVM
The data flow in MVI is unidirectional, with the view sending intents to the model and the model emitting new states that the view then notices.	MVVM, on the other hand, allows bidirectional data binding between the view and ViewModel.
In MVI, the model's state is typically immutable. Each intent triggers a new state that is calculated by the model.	In MVVM, the model's state can be mutable, and changes in the model are automatically propagated to the view.
MVI places a strong emphasis on capturing user intents as distinct actions.	MVVM focuses more on exposing data and commands from the ViewModel to the view.
MVI is very testable due to its unidirectional flow and focus on pure functions. By supplying specified intents and confirming the resultant states, unit testing the behavior of the model is made simple.	MVVM also allows for effective testing, but the bidirectional data binding and view updates can introduce some complexity in unit testing.

Conclusion

- MVVM separates the concerns of UI, business logic, and data by introducing the ViewModel layer, which acts as a mediator between the View and the Model.
- The View is responsible for creating the user interfaces and displaying data to the user. It interacts with the ViewModel to retrieve data and handle user interactions.
- The ViewModel holds the business logic and exposes

data to the View through observable properties or LiveData. It updates the View with the latest data and handles user actions.

- The Model encapsulates the domain and data model of the application. It includes the business logic, data access, and validation rules. The ViewModel interacts with the Model to retrieve and update data.
- MVVM promotes low coupling and high cohesion between components, making the codebase more modular, maintainable, and scalable.
- There are different ways to implement MVVM in an Android project, such as using the DataBinding library provided by Google or combining MVVM with other tools like RxJava for reactive programming.
- MVVM is especially beneficial for Android app development due to its low dependency on Android APIs, support for complex XML layouts, and improved unit testability.