International Research Journal of Engineering and Technology (IRJET) e-ISSN: 2395-0056
Volume: 07 Issue: 09 | Sep 2020 www.irjet.net p-ISSN: 2395-0072

Comparative Study on Android Design Patterns

Dimple Panchal[1]

[1]PG Student, Vivekanand Education Society's Institute of Technology, Dept. of MCA, Mumbai, India.

Abstract - Android is a very popular mobile platform, and has managed to take over the majority of the mobile market. It is true that there are comprehensive studies in the area of design patterns in several object-oriented languages such as Java, C# and C++. Two main leaders that represent the mobile market are : Apple (IOS) and Google (Android). Android Mobile platforms completely depend on the developers use and experience. Android Design patterns in software development have helped in improving software quality. The demand for mobile application development is high and is increasing with time. In order to stand out, a mobile application should be cost-effective and should be of good quality. The goal for improving maintainability, the developer begins to apply various design patterns such as MVC, MVVM, and MVP to the development of Android-based mobile applications. Traditionally, software developers utilize a set of design patterns to foster reusability and better software design. Mobile apps now target domains that are critical such as health, banking, m-payments.

Keywords- Mobile Application Development, Android Design Pattern, Android Apps, Maintainability.

I. INTRODUCTION

The Mobile Market has grown in recent years. Compared to computer programs, mobile applications often have limited functionalities, shorter shelf and lower price. New applications should be developed fast to be cost-effective and updated often to keep users interested. The quality of the application should not be neglected, as mobile users are very difficult and competition is stiff. Android development is the process in which new applications are created for devices running on various Android operating systems. Android-based applications can be written in Kotlin, Java and C++ languages using the Android Software Development Kit (SDK), while using other languages is also possible. Some Programming languages and tools allow cross-platform app support that are for both Android and IOS. SDK includes a comprehensive set of development tools. These include a debugger, libraries, documentation and tutorials. For Instance, Every car requires a powerful engine; every house requires a solid base, in a similar way every software development procedure requires its design pattern. To make it happen, we make use of different technologies and architecture patterns. The three types of most commonly used architectural UI design patterns are MVC, MVP, and MVVM. These design patterns play a significant role in developing an application which is easier to test, maintain and facilitate reusable object- oriented development. These architecture patterns are designed to moderate the complex codes and make the UIcode cleaner and manageable.

II. RELATED WORK

Many related works have been done to prove the benefits of various non-architectural design patterns in the development of software that uses the service, and the results show improved performance post-implementation of the design pattern. Other studies perform comparisons M-V- Design Pattern and produce recommendations development patterns according to the character of the existing architectural design patterns. Then there are also studies that discuss the implementation of the design pattern clearly to a program code. In this study, we concentrate on evaluating the use of software design patterns on the development of Android based mobile applications using anti pattern measuring the level of carried out before and after the implementation of the Model-ViewPresenter, Model-View- Controller and Model View View-Model. This study also makes comparisons between the three model architectural patterns. We do need the architectural design patterns so that everything should be organized in a proper way. By following a proper architecture we do get simplicity, Testability and Low-cost maintenance.

III. ARCHITECTURAL DESIGN PATTERNS

A Design pattern is like a model that has been created to solve a problem that occurs concurrently, thus allowing developers to follow a structure so that they do not have to solve it from scratch, but rather by guiding themselves on something already existing or previously tested. An architecture pattern allows us to define a guide for the 'architecture' of a software system, making it scalable, maintainable and testable. Differing from design patterns, these have a major abstraction level. There are 3 different Architectural design patterns:

1. **MVC (Model View Controller)**

2. **MVP (Model View Presenter)**

3. **MVVM(Model View View-Model)**

IV. IMPLEMENTATION

MVC (MODEL VIEW CONTROLLER)

MVC is the first and foremost architectural pattern specially designed for web applications and introduced in the 1970s. MVC lets you build an foremost architectural pattern specially designed for web applications and introduced in the 1970s. MVC lets you build an application that eases the efforts to test, maintain and extend the application. During a traditional software development procedure, we write relevant code or use user control to form the view a part of the definition class. This procedure increases the size of view class among business operations, data binding logic, and UI. Thus, the MVC architecture pattern is designed to reduce the code size and make a code –good code. Cleaner and easily manageable.

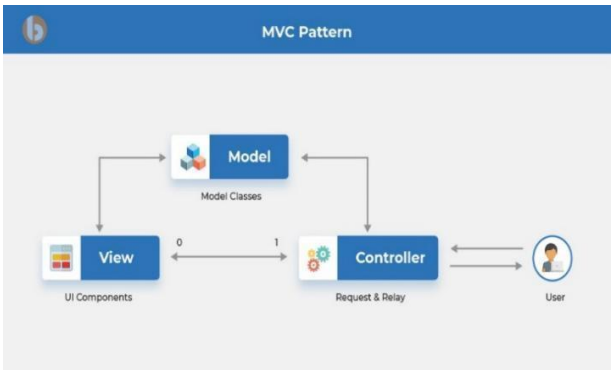

Fig - 1: MVC (Model View Controller)

Model : Contains models of business logic, i.e. the classes that will model the solution for the problem and objective of the application.

For Instance, if you are developing an app for a library, the 'Book', 'Author' and 'Publishing House' classes should be in the package. All of them should be independent from the android component. The data layer, liable for managing the business logic and handling network or database API.

View : View is an interface that is going to be implemented by all the UIviews, that's it. Activities, Fragments, Adapters, Dialogs, etc. All of these are in this package and are responsible for displaying the data to the user. View is just a visualization from data to model.

Controller : Controller is responsible for 'listening' user inputs and updating the model and the view. Normally, the view and controller are involved in the same java class, Imean, an activity accomplishes the functions as a view and controller at the same time because it is listening(for example) user clicks and shows answers for them. The best idea is to have the controller in a separate class.

MVP (MODEL VIEW PRESENTER)

The MVP design has such a ton of likeness to the MVC design. In this design "p"stands for the moderator. The viewmanages also show the page controls. The Presenter is responsible to address all the UI occasions on sake of the view. It gathers contributions from the clients at that point and continues the information from side to side the Model that changes the outcomes back to the View. The Presenter basically performs from the rationale end for signals, for example, press a catch or coordinate streets through route. MVP design is typically performed with windows structures applications and .NET Web Forms. This example is undifferentiated from the MVC design during which the regulator has been supplanted by the moderator. This structure design parts an application into three principle viewpoints : **Model**, **View** and **Presenter**.

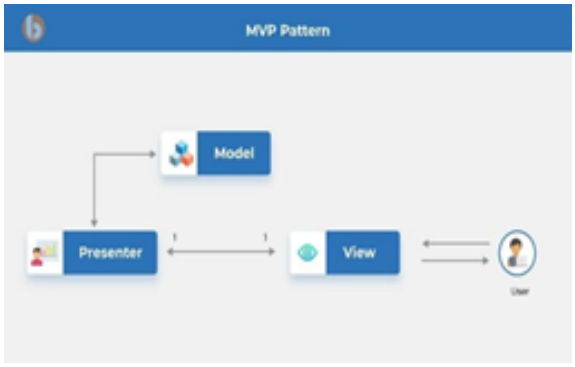

Fig - 2: MVP (Model View Presenter)

Model: Model speaks to a lot of classes that portray the business rationale and information. It moreover characterizes business rules for information implies how the data are regularly changed and controlled.

View: The View speaks to the UIcomponents. It is just subject to showing the data that is gotten from the moderator in light of the fact that the outcome. This additionally changes the models into UI. In run of the mill executions the view segments in MVP sends out an interface that is utilized by the Moderator. The moderator utilizes these interface strategies to control the view. Model strategy names would be: **showProgressBar**, **updateData**.

Presenter: The Presenter is at risk for taking care of all UI occasions in the interest of the view. This gets contributions from clients through the View, at that point measures the client's information with the assistance of Model also, passes the outcomes back to the View. Not at all like view and regulator, view and moderator are totally decoupled from each other and conveyed to each other by an interface.

MVVM(MODEL VIEW VIEW-MODEL)

MVVM has been characterized by MVC. MVVM design upholds two official views between View what's more, View-Model. It permits programmed change inside View-Model to the View. As a rule, the view-model utilizes the eyewitness example to make changes in the view- model to the mode. MVVM has three key parts that are Model, View, View-Model.

Model: The Model describes a lot of classes to depict business rationale. It additionally plots the business rules for information on how information can be taken care of or changed. It handles the SQLite Database, Firebase and Shared Preferences.

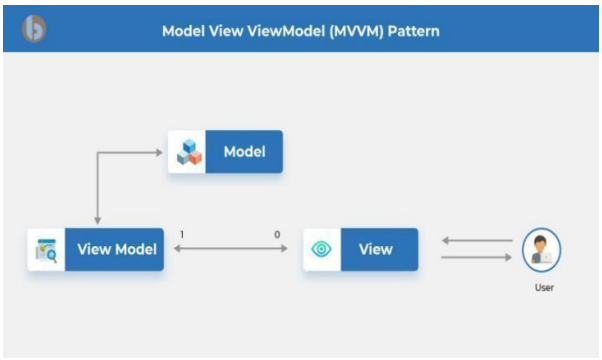

Fig - 3: MVVM (Model View View-Model)

View: The View represents UIcomponents such as JQuery, HTML, CSS, and so on. The View is responsible for showing the information which is recovered from the regulator as a result. It likewise changes the model into the UI. It is liable for taking care of Menus, Permissions, Occasion Listeners, Toasts, SnackBars, Starting exercises, working with Android View and Gadget.

View-Model: The View-Model is for introducing capacities, techniques, and orders to maintain the condition of view, work the model and enact the occasions in the View itself. It is liable for Uncovering information, Input approval, Handling perceivability, Executing calls and techniques to show furthermore, see. The view model should just figure it out the apparatus setting.

The application setting can:

1. Start helping.

2. Tie to help.

3. Send a transmission.

4. Register a transmission collector.

5. Burden asset esteems it can't:

6. Show a discourse.

7. Start a movement.

V. APPLICATION AND MERITS/DEMERITS

MVC is more of an architectural pattern, but not for complete application. MVC mostly relates to the UI / interaction layer of an application.

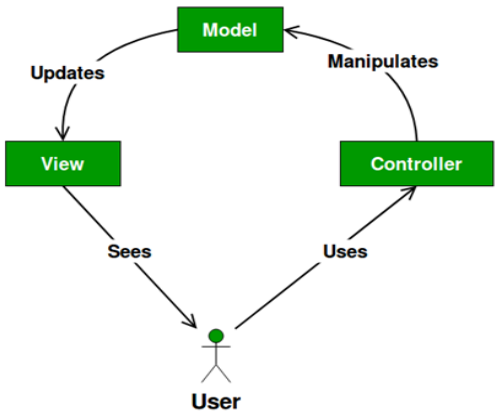

Fig - 4: MVC Flow

The Controller and therefore the View depend upon the Model: the Controller to update the info, the View to urge the info. But, most important for the desktop and Web developers at that time: the Model was separated and could be tested independently of the UI. Several variants of MVC appeared. The best-known ones are associated with whether the Model is passive or is actively notifying that it's changed. By applying the MVC pattern, you permit the bulk of Android developers to readily understand your code. You also make it easier to reuse your code later. For example, you'll export your application template as a library in order that another developer can use it in their application approximately that you simply can use it in your next app.

Merits:

1. Development of the application becomes fast.

2. Easy for multiple developers to collaborate and work together.

3. Easier to update the application

4. Easier to Debug as we have multiple levels properly written in the application.

5. The Model classes don't have any reference to Android classes and are therefore straightforward to unit test.

Demerits:

1. The View depends on the Controller and the Model.

2. Due to the large code, the controller is unmanageable.

3. Hinders the Unit testing.

4. Increased Complexity.

The **MVP** design pattern is preferred over MVC when your application needs to provide support for multiple user interface technologies. MVP has some risks the presenter is attached to the view forever. The View is an activity which can leak the activity and update the activities that already died.

Merits:

1. It's very understandable and easier code maintenance because it separates application's model, view and presenter in layers.

2. View is pretty light, as it only does the visualization as well as it is simple to support (in other words, no more thousands of code lines)

3. Better testing possibilities, as the whole business logic is separated from the UI and the data is simpler to mock.

4. One may painlessly change the framework, as all ties are described in interfaces.

Demerits:

1. As each interface covers interaction to the tiniest detail, it could result in numerous methods.

2. Code size is quite excessive.

3. Huge amount of interfaces for interaction between layers.

MVVM User interacts with the View. There is a many-to-one relationship between View and View-Model means many views can be mapped into the View-Model. Supports two way data binding between View and View- Model. View features a regard to View-Model but View-Model has no information about the View. The figure above shows the architectural components for the MVVM.

Merits:

1. Using the official Google library that assures a proper generation of components.

2. Review at compilation stage.

3. No need for dozens of findViewById, setOnClickListener or similar code.

Demerits:

1. Similarly in bigger cases, it can be hard to design the View-Model.

2. Debugging would be a bit difficult when we have complex data bindings.

VI. COMPARISONS /ENHANCEMENT FOR ARCHITECTURAL DESIGN PATTERNS

When implementing MVC on Android platform things get tricky. To start out the logic finishes up within the Controller. Controller here contains all the logic. Controller has responsibility for what's displayed on the screen.

For simple screens, this will be manageable but as features get added, files will continue to grow. View layer concerns itself with the views. It doesn't realize the database and network layers. Moreover, it's difficult to check components once they are tightly coupled. You ought to test the code that exists within the controller. Much of the code under test will find yourself wanting to run Android SDK components, like activities, which require a full running device or emulator. Basic unit testing won't help; you'll need to use expensive instrumented unit tests to check the essential parts of the app. One alternative to the issues presented by MVC is to decouple a number of the parts from one another. MVP is an architecture pattern that you simply can use to affect a number of the shortcomings of MVC, and may be a good alternative architecture. It provides a simple thanks to believe the structure of your app.

One alternative to the problems presented by MVC is to decouple some of the parts from each other. MVP is an architecture pattern that you can use to deal with some of the shortcomings of MVC, and is a good alternative architecture. It provides modularity, testability and more cleaner and maintainable codebase.

In MVC, the User input is directed to the Controller first, not the View. The input could be coming from the user interacting the page but it could even be from simply entering a selected URL during a browser. There's a many-to-one relationship between the Controller and therefore the View. That's because one controller may select different views to be rendered and support the operation being executed. There's a 1 way arrow from Controller to the View because the View doesn't have any knowledge or regard to the Controller. The Controller does pass back the Model, so there's knowledge between the View and therefore the expected

Model has passed there too, but Controller serves it up.

In MVP, the user input begins from the View, not the Presenter. There's a one-to-one mapping between the View and therefore the associated Presenter. The View features a regard to the presenter. The Presenter is additionally reacting to events being triggered from the View, so View is related to it. The Presenter updates the View supporting the actions it performs on the Model but the View isn't conscious of it. In MVVM, the input begins from the View not the View-Model. While the View holds a regard to the View Model, the View Model has no information about the View. This is often why it's possible to possess a one-to-many mapping between various Views and one View Model. The View has no idea about the Model during this pattern. There's rich communication between the View and therefore the View-Model for the info binding process.

PATTERN	Dependency on Android APIS	XML Complexity	Unit Testability	Modular & SRP
MVC Controller	High	Low	Difficult	No
MVP Presenter	Low	Low	Good	Yes
MVVM ViewModel	Low or No dependency	Medium or High	Great	Yes

Fig - 5: Comparison

VI. APP USED FOR IMPLEMENTATION

For the implementation of the architectural patterns for instance the MVC and MVVM main difference is that the API used is named. In MVC, The API is named again and again whenever there's an invitation from the User. When the user either turns the screen during a portrait to landscape view the API is named again and again on the request so it becomes difficult to handle the API request in MVC. MVVM, the API is named just one occasion for the landscape or portrait view so it becomes easier to handle the request and makes the code run efficiently. It uses Mutable live data which is employed for observing changes within the view and updating the view when it's active. The appliance is made using the retrofit and Async task. Retrofit is far more popular than volleyball. The parsing is completed using JSON libraries during which you only need to create models for JSON nodes and therefore the library automatically converts the JSON into appropriate objects. Here, we've used the API to fetch the info.

https://www.simplifiedcoding.net/demos/marvel

The above URL is returning an inventory of Marvel Super Heroes. Also uses Recycler view to display the list of heroes and it's the scrollable view and Glide which loads the image from the URL. The advantage of doing this manner is, if our activity recreates then it'll not fetch the info again making our application more efficient. In MVP, the presenter is employed to urge data from the API and send it to the View. The API is named again and again on the request of the user. Here we've used an equivalent example and an equivalent API for the implementation. The figure below shows the View of the application.

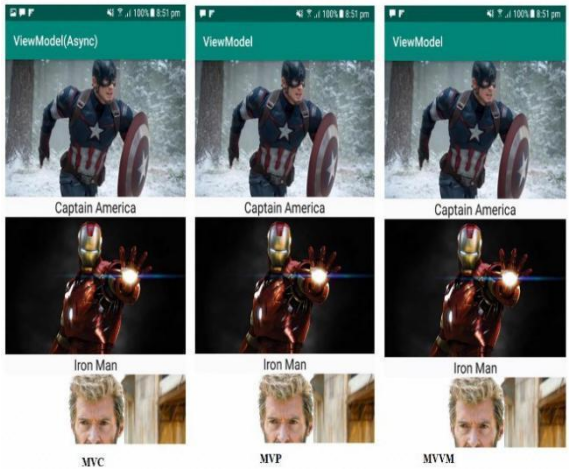

Fig - 6: Example

VIII. CONCLUSION AND FUTURE WORK

Using a design pattern, on the opposite hand, will increase the amount of lines of code and energy needed to write down code but on the opposite hand decrease the complexity of program code. This causes an insignificant increase in maintainability. However, within the end, this study complexity of program code. This causes an insignificant increase in maintainability. However, within the end, this study proved empirically although Line of code is increasing in order that it needs more effort to write down program code, use of Design pattern proved to extend maintainability and modularity of android based mobile application development. This research is predicted to be a source of more in-depth knowledge on the utilization of design patterns within the world of mobile application development. In fact, there's no specific architecture implementation you want to use. All depends on your business requirements, time, software team, resources, etc. i prefer to use MVP once I don't want to write down an excessive amount of code; on the opposite hand, i prefer implementing Clean Architecture when the project is big and changes might be requested within the future.

REFERENCES

[1]https://www.simplifiedcoding.net/android-Viewmodel-using-retrofit/

[2]https://www.simplifiedcoding.net/demos/marvel/

[3]https://www.bacancytechnology.com/blog/mvc-vs-mvp-vs-mvvm

[4]https://android.jlelse.eu/architecture-patterns-in-android-abf99f2b6f70

[5]https://upday.github.io/blog/model-view-controller/

[6]http://geekswithblogs.net/dlussier/archive/2009/11/21/136454.aspx

[7]https://ieeexplore.ieee.org/document/522317

Design patterns in software development by IEEE

[8]https://ieeexplore.ieee.org/document/522317

[9]https://medium.com/@ankit.sinhal/mvc-mvp-and-mvvm-design-pattern-6e169567bbad