**SOFTWARE DEVELOPMENT FOR MOBILE DEVICES**

COS80019

ASSIGNMENT 11 – Research Report

**Android Architecture Pattern: MVC, MVP and MVVM**

**Student name: Duy Thuc Pham**

**Student number: 101767225**

**ABSTRACT**

# INTRODUCTION

Mobile applications have become an essential part of human being nowadays. According to Statista, the number of applications in the Google Play Store was 2.6 million in March 2018 [1]. However, the rapid change of new technology, as well as user requirements, have led to the need for the improvement of architecture pattern, which may help developers quickly adapt to the alternation and reduce the development time. Many architecture patterns have been applied to the mobile application throughout the year, but MVC is considered to be top the most famous in terms of "separate of concerns" and easy for implementation. Also, it has been used as the default architecture pattern for both iOS and Android. In Android development, though it is quite popular, it is concerned for the extensibility and testability [2]. Furthermore, the various Android developers have applied MVP or MVVM instead of MVC as they consider MVP and MVVM are more extensible and testable [2]. Therefore, this paper aims to evaluate MVC, MVP and MVVM to justify the most appropriate architecture to apply in Android development with respect to maintainability, reusability and testability.

In this research, the brief information of MVC, MVP and MVVM will be introduced. Thereafter, it will illustrate the design of these three architectures in Android development. Then, these architectures will be implemented with the given scenario by following the design section. Finally, the evaluation for these architectures will be conducted to understand which architecture is sufficient with regards to maintainability, reusability and testability.

# METHODOLOGY

## Background

### MVC

MVC is a well-known architecture in software development. By following “separate of concerns” principle, it divides the program into three components: model, view and controller. Additionally, it leverages the benefit of “separate of concerns” by decoupling significant features and hence the code is loose coupling and high cohesion.

**Model**

Model represents the data layer which is used to hold the application data and manage the data logic [3].

**View**

View is the UI layer which provides the user interface to the user based on the data model.

**Controller**

Controller is the logic layer which considers as the mediator between the model and view. It takes responsibility for updating the data model and notifying to the view to update the user interface.

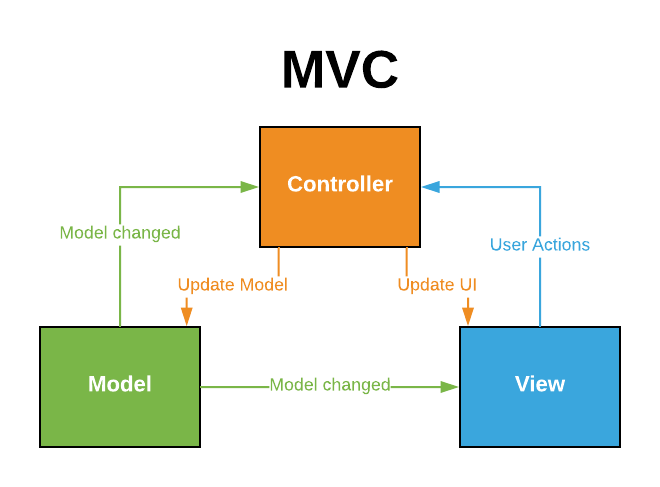


Figure : MVC pattern

### MVP

Theoretically, MVP pattern is almost the same as MVC pattern by replacing the Controller to the Presenter and also, they both have three separate components.

**Model**

Just like the model of MVC, it is responsible for managing data.

**View**

View displays the data to the user interface and notifies the user actions to the Presenter.

**Presenter**

Presenter manipulates the user actions from the View. Then, it updates the data of the Model and notify the updated information back to the View. It is slightly different from the Controller as the Presenter does not have a reference with the View and vice a versa, but instead they use the interface to communicate with each other [4].

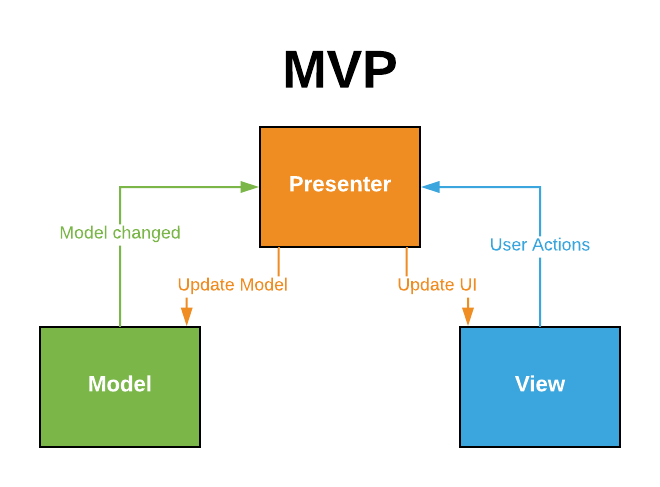


Figure : MVP architecture pattern

In addition to the difference between MVC and MVP, in MVP, the Model and View are entirely decoupled. The Presenter is responsible for controlling the user events and the data model.

### MVVM

Model-View-ViewModel (MVVM) is an architecture pattern which was announced in 2005 by John Gossman [6]. The aim of this pattern is to optimize the event driven programming by providing the data binding between View and ViewModel [5].

**Model**

The Model is similar to these patterns above, which controls the business logic of the data layer.

**View**

The View retrieves the user actions and sends notification to the ViewModel.

**ViewModel**

The ViewModel acts as an intermediary in this pattern. It is on the one hand, provides methods, properties to the View with the purpose of maintaining the state of it. On the other hand, it manages the Model based on the user actions receiving from the View and invoking the view to update the UI.

****

Figure : MVVM architecture pattern

Unlike MVP, while the Presenter must inform to the View what to show, the MVVM supports the data binding mechanism in which the View can bind to. Therefore, it eliminates various interfaces in MVP pattern. Moreover, as in the Figure 3, in MVVM, many View can bind to one ViewModel in a many-to-one relationship [5], which decouples both components because the ViewModel does not need information from the View.

## Design and Implementation Architecture in Android

In this part, the appliance of MVC, MVP and MVVM in Android are illustrated based on an example application. In this research, the example of application is to convert inch, feet and mile into centimeter which has been already implemented in task 3.2P. The research starts with the design and implementation for MVC, MVP and MVVM respectively.

### MVC

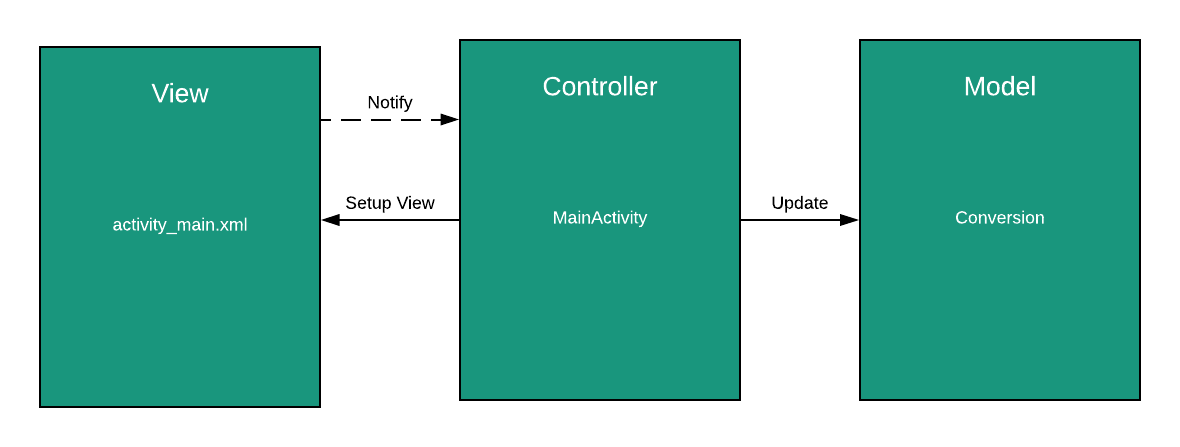


Figure : Apply MVC in Android

In Android, MVC is considered as the default pattern to develop Android application. The View is the layout xml, the Controller is Activity or Fragment and Model is the Java class which presents the data layer of the application. Controller layer seems like the most crucial part of the MVC pattern due to the plenty of responsibilities of it. The figure 5 illustrates how the Controller is implemented in MVC. Firstly, it retains references with view components (e.g. EditText, TextView, ImageView, etc.). Secondly, it manages the model instance so that it can update the data for the model right after receiving the user actions. Thirdly, it handles the user actions from the View layer by implementing the listener event such View.onClickListener, View.onTouchListener, etc. Finally, it manipulates the business logic to display the appropriate UI to the user.

|  |
| --- |
| public class MainActivity extends AppCompatActivity implements View.OnClickListener {   // Establish references with the view layout  EditText edtFeet;  EditText edtMiles;  EditText edtInches;  CheckBox chkboxDisplayMetres;  TextView txtViewResult;   private Conversion conversion;   @Override  protected void onCreate(Bundle savedInstanceState) {  super.onCreate(savedInstanceState);  setContentView(R.layout.*activity\_main*);  initializeUI();  // Instantiate the model  conversion = new Conversion();  }   private void initializeUI() {  edtInches = findViewById(R.id.*edtInches*);  edtFeet = findViewById(R.id.*edtFeet*);  edtMiles = findViewById(R.id.*edtMiles*);  txtViewResult = findViewById(R.id.*txtViewResult*);  chkboxDisplayMetres = findViewById(R.id.*chkboxDisplayMetres*);   Button btnConvertMiles = findViewById(R.id.*btnConvert*);  btnConvertMiles.setOnClickListener(this);   }   */\*\*  \* This method uses to check whether these edit text have value or not  \** ***@return*** *\*/* private boolean isValid() {  return edtInches.getText().toString().length() > 0 && edtMiles.getText().toString().length() > 0 && edtFeet.getText().length() > 0;  }   @Override  public void onClick(View view) {  String result;  String metres = getResources().getString(R.string.*metres*);  if (isValid()) {  conversion.setData(Double.*valueOf*(edtInches.getText().toString()) ,Double.*valueOf*(edtFeet.getText().toString()), Double.*valueOf*(edtMiles.getText().toString()), chkboxDisplayMetres.isChecked());  if (chkboxDisplayMetres.isChecked()) {  result = String.*format*("%.2f %s", conversion.toValue(), metres);  } else {  result = String.*format*("%.2f cm", conversion.toValue());  }  } else {  result = "";  Toast.*makeText*(getApplicationContext(), "You have to enter a valid value", Toast.*LENGTH\_SHORT*).show();  }  txtViewResult.setText(result);  } } |

Figure : Controller in Android

### MVP

The design and implementation for MVP pattern are slightly different from MVC pattern. While the model layer is the same, View and Presenter are now communicating with each other via an interface.

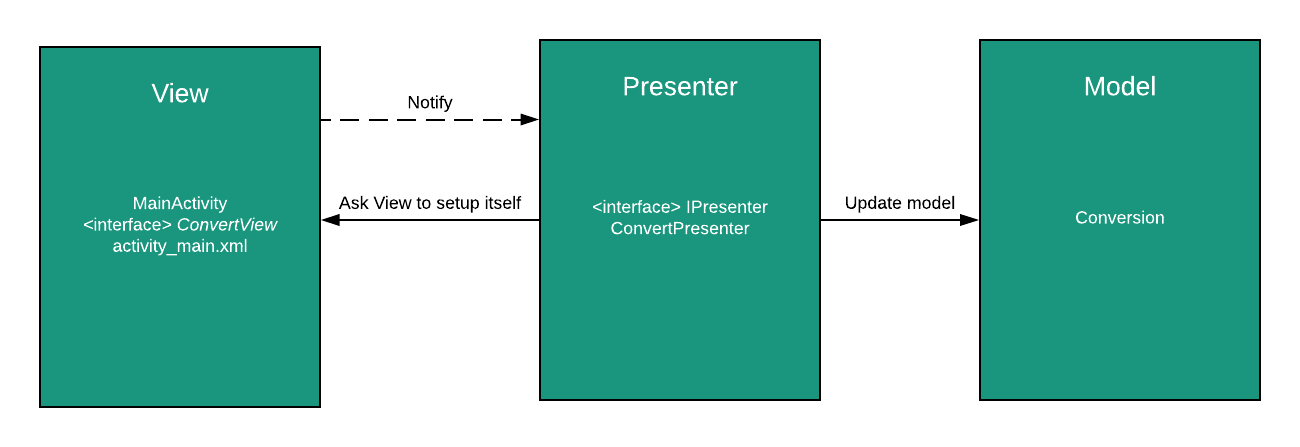


Figure : Apply MVP in Android

### MVVM

# DISCUSSION

|  |  |  |
| --- | --- | --- |
| MVC | MVP | MVVM |
|  |  |  |

# CONCLUSION

# REFERENCES

[1] <https://www.statista.com/statistics/266210/number-of-available-applications-in-the-google-play-store/>

[2] <https://academy.realm.io/posts/eric-maxwell-mvc-mvp-and-mvvm-on-android/>

[3] <https://medium.com/upday-devs/android-architecture-patterns-part-1-model-view-controller-3baecef5f2b6>

[4] <https://android.jlelse.eu/android-architecture-2f12e1c7d4db>

[5] <https://upday.github.io/blog/model-view-viewmodel/>

[6]<https://blogs.msdn.microsoft.com/johngossman/2005/10/08/introduction-to-modelviewviewmodel-pattern-for-building-wpf-apps>

# APPENDIX