MVC Vs. MVP

Survey of Programming Languages
ITCS-4105/5102
Final Project Report

28 June 2021

Austin Lowery

1. Introduction

At times, it can be difficult to organize a large assortment of code. It isn’t always enough to program and work through problems as you go along, which is why UI Design Patterns were put into practice. MVC, one of the oldest design patterns (Likness & Garland 353), was created as a way to address the problem of large and complex data sets under the control of a user (Reenskau 1). To further address the separation of concerns in the MVC design pattern, the controller was replaced with a presenter which coordinates between the user, the model, and the view; this new design pattern was called MVP (Likness & Garland 354).

The MVC pattern is unique in that its controller handles the connection between the model and the view (Banas). The MVP pattern instead has a connection between the presenter and its view (Rasmusson). With these two differing patterns, it brings me to wonder which might be the better of the two.

1.1 Purpose

As a programmer, I want to be able to understand the best coding practices and methods. Implementing UI design patterns seem to be a part of that and are something I have gathered very little experience in. For this reason, the topic of this project is to compare and contrast the MVC and MVP design patterns. I am interested to see how difficult these design patterns will be and how each language I end up using contributes to its difficulty. At the end, I am hoping to walk out of this with a reliable design pattern for my own personal use.

1.2 Scope

The only software products used were Python, Java, and their packages. These were used without the need for external sources because of my experience in both this class and other classes. However, my complete understanding of MVC and MVP can be attributed to Derek Banas and Jonathan Rasmusson respectively (Banas)(Rasmusson).

1.3 Definitions

- MVC: Model, View, Controller
- MVP: Model, View, Presenter

1.4 References

Banas, Derek. “MVC Java Tutorial”, YouTube, 21 Feb 2013. Web.

Likness, Jeremy & Garland, John. "Programming the Windows Runtime by Example", Addison Wesley Professional, Boston, MA, 22 May 2014. Web. Obtained on Google Books.

Rasmusson, Jonathan. “Model View Presenter”, YouTube, 10 May 2020. Web. Obtained on YouTube.

Reenskau, Trygve M. H. "MVC Xerox Parc 1978-79" University of Oslo, Oslo, Norway, 22 Mar 1979. Web. Obtained on universitetetioslo.no.

1.5 Overview

The overall difficulty of the project was not too terrible. There were plentiful videos on both design patterns and this aided in my ability to find the perfect one for me. The videos I watched for each one were in a different programming language so I had to translate the code into the one I would be using. This was somewhat difficult for the MVP design pattern as I did not know Swift. This was the hardest part of the project which tells a lot on just how useful and easy to understand these two design patterns are.

The application itself was simple as it was designed that way intentionally. With this project, I gained a better understanding of both the MVC and MVP design patterns.

2. General Project Description

A basic application was made for the purposes of analyzing each design pattern as opposed to the application itself being the product of this project. For this reason, a basic meal cost calculator was made for each design pattern. Each design pattern was considered and compared between one another on the basis of difficulty, simplicity to use, and easiest to understand. The difficulty of each language used will also be expressed in an attempt to consider possible bias.

2.1 Project Perspective

The view I had in mind for this project was not anything too fancy. As mentioned, it was going to be a simple application meant only to compare the two design patterns. I would say that this goal was achieved.

2.3 Specific Goals

The first goals of this project were to create a single application implementing these different design patterns. Of course, this would require researching the two design patterns which themselves may be considered a goal. Finally, the goal of deciding which design pattern I had preference over was the main goal of this project.

2.4 Overview of Programs Related to Specific Goals

The decided program of choice was a basic meal cost calculator which took into account the cost of the meal, the tax of the meal, and the tip given to the waiter. This was accomplished using simple math and interfaces. While very simple in concept, it was enough to compare the design patterns' usefulness. The decision of which I had preference over was ultimately achieved which will be explained later in this paper.

2.5 Assumptions and Dependencies

The only assumption I had going in is that I would enjoy the MVC design pattern based on two reasons. The first reason being I had the most experience with them and understood it the most. The second reason was that it seemed the most simple out of the three design patterns which I normally enjoy more over efficiency.

3. Programs Developed

One program was developed, but it was changed and adjusted to fit the two design patterns. The program itself was not the goal of this project, but the design pattern implemented into it.

3.1 Program Specification

3.1.1 User Interface

The user interface was basically the same for each app with minimal visual differences. It contained basic labels for each text field, entry boxes to input values i.e. cost, tax, and tip, a button to calculate the total cost, and a result box to display the total cost.

3.1.2 Program Logic and Code Explanations

The MVC application was designed in python using tkinter. It uses three separate files which are the model, view, and controller files for the application. The model handles all the calculation work, the view handles all the ui components of tkinter shown on screen, and the controller are all matters of input needed to communicate between the view and the model.

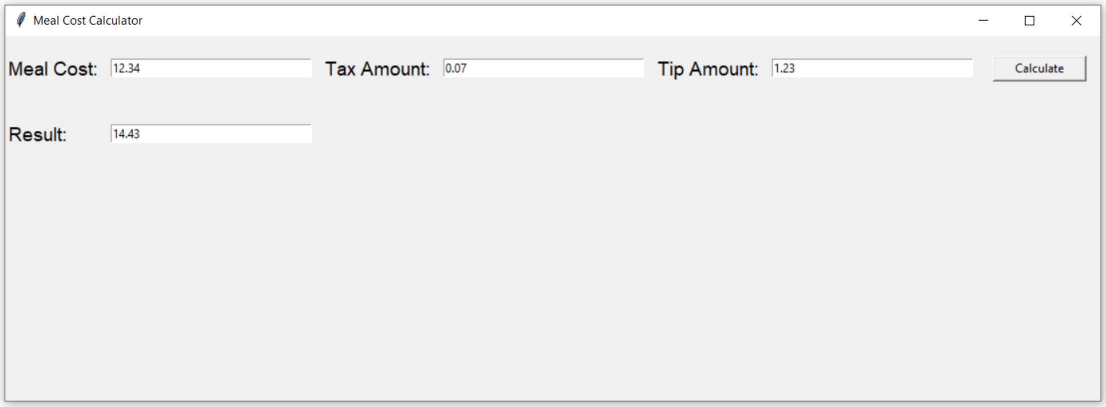
The MVP application was designed in Java using android studio. It is contained within a single file which holds the model, view, and presenter classes. The model and view act in the same way as it does in MVC. The presenter is much like the controller except it has further separation of features and handles some of what an MVC model might rely on the model for. It uses a talker to translate a response from the presenter to the view.

3.1.3 Inputs/Outputs

The inputs were the cost, tax, and tip value boxes. The output was the total cost.

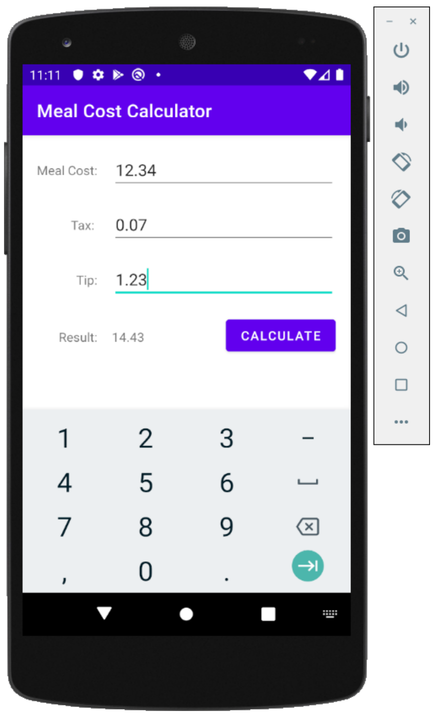
3.1.4 Program Results

MVC Design Pattern



Meal Cost: 12.34 Tax Amount: 0.07 Tip Amount: 1.23

Result: 14.43


Meal Cost Calculator

Meal Cost: 12.34

Tax: 0.07

Tip: 1.23

Result: 14.43

MVP Design Pattern

4. Discussion and Conclusion

After analyzing each of these design architectures, I believe that the one that I would prefer would depend on the circumstance. However, for general purposes, I would consider the MVC architecture the winner. If I were to be a member of a team of programmers, the MVP architecture may be the better option as it is easier to read.

4.1 Simplicity

I found that I had a much easier time understanding the MVC design pattern, possibly due to my past experience in it. Despite this, I still believe that even without this experience I would find it more simplistic. MVP seems to divide its features in a way that is simple to understand after getting completed, but more difficult in practice to achieve.

4.2 Readability

This may be due to Java looking cleaner to me compared to python, but the division of all the elements in MVP appear to more easily convey the inner workings of what is happening in the program. I can look at the presenter and see pretty well what is happening with the program. The values being held in the model also neatly keep it more simple to understand. The MVC application appeared less readable to me, although this may be because of a lack of polish and understanding of tkinter. Since the MVP program keeps a lot of the view information inside the xml files, it is difficult to gauge how big of a difference the views are between the two design patterns.

4.3 The Languages

I have always had a preference towards Python over Java, which may have played a role in my individual preference of MVC. Java, to me, is more for group developers so this may be why I prefer MVP in a group setting. I will have to continue using these design patterns in different languages to see if my opinion changes in the future.

4.4 Further Thoughts

There was likely more that could have been explored had I some sort of base application to go off of. Something more complex may have given me more of an idea of what my true feelings on these design patterns would be. It is possible that the simplicity of the application hindered my ability to give a proper opinion on it, but for now, I am happy with the results.