UDC: 004.414.22.054:004.2
Professional paper

MVC VS MVVM VS MVP: WHICH ARCHITECTURAL PATTERN SUITS MODERN APPLICATIONS BEST?

Alban IBRAIMI, Agon MEMETI [0000-0002-6824-3856]* , Florim IDRIZI[0000-0001-7514-3282]

Department of Computer Sciences, Faculty of Natural Sciences and Mathematics, University of Tetova, Tetova, North Macedonia
{a.ibraimi241970; agon.memeti; florim.idrizi}@unite.edu.mk

Abstract

The exponential growth of software development in the past decade has made architecture decisions as key determining factors of scala-able, maintainable and testable applications. There are a lot of architectural patterns themselves and although they are not alike with respect to the strength and weakness, they also vary in terms of form, components responsibility ánd fit within real-world projects which can be fruitfully beneficial learning from each other for each's development work. A comparative case study of three popular architecture patterns, MVC, MVVM and MVP is conducted by comparing their theoretical basis, usage in real-world and relative strengths and weaknesses on. NET-based projects. Based on a practical development of three task manager applications, this study has shown us the different impacts of 3rd TMS on software architecture, maintainability and development productivity. The findings are directed towards aiding developers to decide on the most suitable architecture according to specific needs of projects, development context and Maintenance issues in a longer range.

Keywords: MVC, MVVM, MVP, software architecture, architectural patterns.

1 Introduction

The architectural decisions are very important while designing applications nowadays, as they determine the base on which is built scalability, maintainability and testability. Of all the most popularly used patterns that determine user interface logic, MVC (Model-View-Controller), MVVM (Model-View-ViewModel), and MVP (Model-View-Presenter) are perhaps most widely used among all these. These architectures adhere to the ideology of separation of concerns by separating applications into distinct components responsible for data manipulation, rendering the user interface, and handling user interaction. While these have this ultimate goal in common, these differ significantly in a way in which these permit components to communicate and tackle complexities of applications' modern-day requirements.

There are good uses of all three architecture patterns in the .NET world, thus leaving developers free to decide their best-suited method applicable to their applications. MVC was popular in web development because of its slim, controller-based architecture that always segregates view from logic. MVC was made possible by new technologies like Windows Presentation Foundation (WPF) and Blazor, and MVVM was brought into prominence because of two-way data binding and reactive state manipulation. MVP itself was popular in desktop applications and later found its prominence in web development by using ASP.NET Web Forms and Razor Pages, in which its rigid segregation of rendering of UI from the presentation logic can benefit in testability and maintainability.

This work tries to compare these three architecture patterns both in theory and practice by writing the same task manager applications using MVC, MVVM, and MVP on the .NET environment. This paper tries to see how each architecture affects code structure, maintainability, scalability and speed of development and, as a summary of all, better architecture choice by developers.

2 Related Work

Other writers have studied architecture patterns and their usability in various development environments. Flores et al. [1] studied the application of MVP architecture in solution designing of shortest-path problems and MVP's advantages in solving complexity and broadening testability. Sutar and Katti [2] compared MVC, MVP, and MVVM in application in native Android applications and their complexity in application and difficulties in maintainability. While Iqbal et al. [3] provided ontology-based MVC architecture pattern discovery procedures and structural software architecture identification data.

These concepts build a basis on which this work is founded by observing their suitability and use on a day-to-day manner in picking a particular pattern and not picking another while making a choice between different development options as matters concerning .NET applications [4][5]. To experience actual real-world distinctions between MVC, MVVM, and MVP, three C# and .NET-based task manager applications have been developed. These applications were maintained as functional equivalents while their own implementation revolved around displaying, creating, and manipulating product data, and hence creating a common ground on which to compare between the three flavors of architecture. This experiential approach enabled us to conclude distinctions between code composition, testability, maintainability, and programmer experience in general between a pattern and another [7][8].

It concluded with a comparison of the pros and cons of each architectural style. It provides a very organized way of structuring the user input layer, business logic layer and presentation layer in MVC. Controllers for ASP. MVC controllers in ASP. MVC is just some intermediaries (controllers) that react to a request & fetch/send the data from the database you use (CRUD - CREATE, UPDATE, DELETE or DESTROY)[4][5].

The Model-View-ViewModel (MVVM) pattern in this model that was recommended and presented by Blazor to create modern interactive apps brings the following advantages [6]. Two-way data-binding in MVVM significantly reduces the complexity of handling such dynamic changes to underlying data and allows you to build reactiveUIs with ease. Data (and behavior) encapsulation of ViewModel (MVVM, of course) results in a very testable UI and less sneakiness in the coupling of view to business logic. At first glance, especially if you’re a less experienced developer, the MVVM architecture can appear to be overly complex and time consuming, front end development could potentially slow down on account of data binding slowing larger apps. But when used properly the code will be crystal clear, maintainable and of course highly s with testability.

But in MVP architecture, responsibility is divided by only taking UI concerns through a few Presenters [7]. This is why unit work works so well, you can test a presenter (unit) without views. In either case the comparison was successful in that it showed clear strengths and weaknesses for architecture pattern 1 examined here. MVC Smile (From Real MVC Architecture) It is good exercise so you know how to clean. It does a fine job of providing layered inputs between your user business and something that is de-codable as "presentation". ASP. MVC's are a layer between doing what you want to incoming requests, getting some data for it and returning views of something or other. It also encourages testing at the unit level, which is a natural sequitur for webapps. When complexity begins to work its way through your application then controllers appear and they become larger with logic (fat controllers). In addition, a strong coupling between MVC model and view might also introduce some issues in terms of flexibility when developing dynamic high interactive front-ends.

The MVVM pattern itself, such as exposed here in Blazor (MVU in fact), gives some very neat benefits when we write those modern reactive applications. Two way data binding in MVVM is a pleasure when you develop reactive user-interfaces, as soon as there’s any behind-the-scenes change to your bound data, the connected UI elements will react to that automatically.

With MVVM, applications remain testable and the unidirectional relationship between views and business logic doesn't choke on itself.
Even though MVVM presents a learning curve for new users accustomed to other layers of abstraction, and data binding sometimes impacts performance on very large applications, its benefits make up most of its popularity.
Though MVP architecture requires a hard separation of concerns by routing all UI logic through a presenter, this results in unit testing becoming exceedingly efficient by virtue of being able to unit-test presenters regardless of their view. Though MVP's well-drawn lines do enhance maintainability, its wiring between components by hand results in verboseness and development overhead. In new web scenarios, MVP does become a bit dated, by virtue of its poor technology support for new .NET technology. The actual scenarios mirrored that architecture patterns are not just decisive on code architecture, but on development flow, testability, and maintainability of software systems as well[4][5].

3 Research Methodology

To this end, a cross case study has been carried out to compare MVC, MVVM and MVP with respect to their effects on software quality. The methodology was composed of three major components. Firstly we searched the literature for which are the usual criteria applied when comparing SAs patterns such as maintainability, testability, scalability and complexity. These are the criteria we used as a starting point for our analysis.
Second, we developed three equivalent task management applications in C# using. NET 6 platform. All apps were built to offer the same basic features of a TODO management application, such as adding, editing and listing tasks (i.e. not really practical but amazing for our purpose) in strict accordance with each architectural pattern. Development took place in Microsoft Visual Studio 2022, while the MVVM derivative is based on Blazor and the one of ASP. ASP.NET Core MVC for the MVC flavor, and ASP. NET Web Forms to give a view of the MVP variety for that type. This allowed us to ensure consistency by using the same outside factors in all and each implementation.
Lastly, we compared the three applications using our chosen criteria. For each application, we evaluated (a) code organization, (b) ease of extending functionality, (c) unit test coverage, and (d) time performance during typical operations. Results were subject to qualitative comparison and cross-validated with findings in the existing literature. This approach allows the analysis to be replicable and based on empirical evidence rather than purely theoretical derivations.

4 Case Study: Implementation of MVC, MVVM, and MVP in .NET

To go alongside the theory analysis, a case analysis was suggested in practice to compare and contrast the use and influence of MVC, MVVM, and MVP software architecture designs within the .NET environment. Three different task manager applications were designed in C#, and they were designed to use one of the software architecture designs. To demonstrate the capability of those three implementations, examples are given below. The development environment for both applications was Microsoft Visual Studio®2022 on. NET 6.0 with C#. The ASP MVC application was developed with ASP. NET Core MVC, whilst the MVVM solution used Blazor and the CommunityToolkit. MVVM library, and the MVP derived from it was developed with ASP. NET Web Forms. Figures 1–3 display representative screenshots of the applications as they look when unbuilt (project structure view) and running. In all three applications, their main functionality was identical in consideration of having a common base for a comparative analysis.
The high-level goal for this case study was to see how one architecture handles separating out concerns, interacting with users, and responsibilities of applications in a contemporary

development context. Using and assembling the same functionality per pattern made possible differentiable and real-world distinctions between code composition, development complexity, and maintainability and testability on a unit-by-unit basis.

From these case studies, experience was made on strengths and weaknesses inherent to both architectures. The result of constructing these applications not only validates comparative analysis described above in this work, but also provides real-world recommendations for developers creating their .NET developments and targeting a correct architecture pattern.

To complement the theoretical analysis, a case study at a practical level was also conducted through developing three tiny task management applications in C# and the .NET platform. These three applications shared the same high-level functionality but assumed one particular architectural style, that is, MVC, MVVM, or MVP. This allowed all three methods to be compared head-to-head in order to examine how they divided concern, handled programmatic interaction, and structured their codebase.

Under MVC implementation, MVC conventions were adopted to structure the project, dividing it into Controllers, Models, and Views. User requests were handled by controllers, which pushed data towards views. MVC was very structural, but unfortunately it ends up that controllers can get large and bloated as our use cases grow.

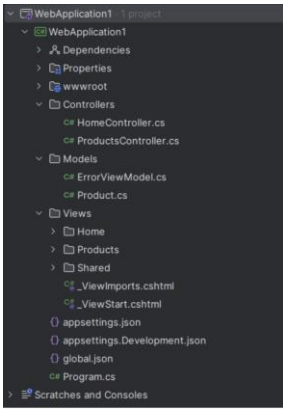

Figure 1. A MVC Project Structure

The MVVM flavor, which included two-way data binding and reactive components, likewise was penned with Blazor. Its hierarchy was formed by Models, ViewModels and Views (where the ViewModel also acted as an in-between layer for data and UI). On successful implementation of the MVVM, writing responsive or interactive UI became much easier, but it had some overheads and learning curve.

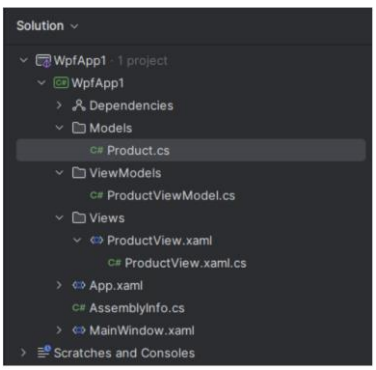

Figure 2. A MVVM Project Structure

Use of MVP, which in ASP. Net Web Forms was of course born and embedded logic into the Views, Presenters and Models. All the business logic was created among presenters, that made

an ability for unit tests and also it solved some front-end decision-making front-back. However, more wiring was required by hand and it was way too verbose than MVVM or MVC.

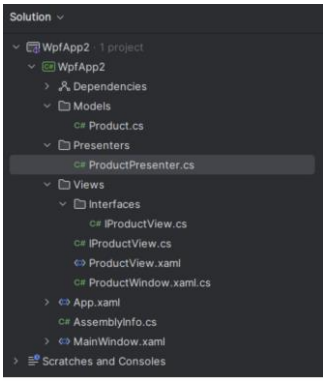

Figure 3. A MVP Project Structure

Therefore, all the concluded examples recognized that architectural decisions have a great influence not only on the code architecture itself but also on its testability, maintainability, and development speed. The MVC starts off sharply but leaves us fat controllers in big projects.
Although MVVM has great abilities for responsive, contemporary user interfaces it seems to be overhead intensive. In concrete: this case study has provided us with data on some essentials we have learned about how such architecture patterns actually stand in practical work on.NET. Architecture always has to be changed for some particular project needs and constraints. Also, in all three approaches, the case study revealed some practical implications of choosing the alternative architecture. On the one side MBA, which had given us fast prototyping and light architecture showed us bloated, unreadable controllers after feature adding.
MVVM gave a very maintainable, interactive solution, which is suitable for the modern web apps, which are Blazor-based, but gave an extra workload through ViewModel encapsulation. On the other hand MVP gave to us ultimate testability, separation of concern, but demanded a very hard manual linking and sounding too verbosely, for example, in two other variants. Large comparisons of development work, code organization, and performance thinking which were done in such practical examples show how architectural decision is implemented in some project needs and team specialization.
These implementations are now available to make possible the replication of our case study. For the ones who are interested we recommend these projects to be downloaded, executed and if required to be expanded for our comparative analysis. This is the evidence to justify that the comparative research was indeed not just a theoretical one but also had practical value based on real implementations.
To evince our analysis, we considered MVC, MVVM and MVP along several dimensions based on literature as well as our own implementation. These parameters are code complexity, scalability, testability, maintainability, learning curve, performance and framework support.
Endorsed as effective and clean in simple domains, MVC was designed for small projects and applications; however, controllers quickly became bloated to the point that maintenance of them has fallen aside. The spam dating free lunch couplet between view and model also prevented the flexibility on highly-interactive systems.
ViewModel ViewModels and two-way data binding, health testing into Blazor and WPF MVVM provided stronger maintenance-ability plus test-ability with its support to ViewModel.Models and two way databinding. Developers had a higher learning curve and performance overhead on large datasets, but it supported scaling well.

MVP provided a great testability story thanks to the separation of presenters from views, but resulted in a big manual wiring effort which did slow us down. It has also become disused with modern. NET frameworks.
There were some next-level things behind the comparatives that could have only been taken apart by having gone through the act of doing them. And the features ain't wired-up the same way between patterns - MVC let you prototype 'pretty' fast, MVVM required a lil bit of setup but MVP was just fucking delayed by those components wiring up. These implementational considerations were orthogonal to the structural properties, but it provided an interesting view on what was actually going on with how each pattern affected productiveness in the day-to-day operation.

5 Conclusion

As far as Quality, Preservation and Portability of the current system is concerned then it will be taken care of by Design Patterns. MVVM is the architecture you should use when you care about responsiveness and maintainability of your UIs, but are willing to suffer a steep learning curve and increased complexity in order to adopt this pattern.
When we start to think about a big project with responsive UI's and large enterprise level apps, the first word that comes into our mind is MVVM…this implies to us that you are in search of maintainability and loose coupling if not even more especially if we are talking about this subject. NET. Also this study tells that MVVM is also pushed by few latest trends (not knowing: Blazor etc) as the alternative way to handle a state and dev reactive UI. These new MVVM features would allow the trend of MVVM possibly here and there taking over enterprise applications (whilst it is still not the focus of this work). applications using MVP with a decreasing need for MVP in time. This paper attempts to harmonise theoretical findings with practical implementation on the one hand and an insight beyond sterile experimentation of how architecture patterns are used in the real world' in the course of 'real-world' development on the other.
Overall, I would say that the architecture applies to you based on application analysis and team skill, footsteps of future supportability, so these pradigms can withstand better for evolving needs wanting to build software now.

References

[1] Flores, H., De Jesús, J., Olague, H. (2020). MVP Architecture and Design Patterns Applied to an Optimal Development of a Software Used for Shortest Path Problem Study. International Journal of Applied Engineering Research, 15(5), pp. 519–526.
[2] Sutar, A., & Katti, R. (2019). A Comparison of Android Native App Architecture: MVC, MVP, and MVVM. International Research Journal of Engineering and Technology (IRJET), 6(6), pp. 2574–2578.
[3] Iqbal, A., Hussain, M., & Rasheed, H. (2018). Identify MVC Architectural Pattern Based on Ontology. International Journal of Advanced Computer Science and Applications (IJACSA), 9(4), pp. 393–398.
[4] Fowler, M. (2002). Patterns of Enterprise Application Architecture. Addison-Wesley, Boston, MA.
[5] Soni, D., & Godbole, K. (2018). Comparative Study of MVC, MVP and MVVM Patterns. International Journal of Computer Sciences and Engineering, 6(3), pp. 465–469.
[6] Freeman, A., & Sanderson, D. (2020). Pro ASP.NET Core 3: Develop Cloud-Ready Web Applications Using MVC, Blazor, and Razor Pages. Apress, Berkeley, CA.
[7] Eckstein, J. (2006). Agile Software Development with MVP. IEEE Software, 23(6), pp. 80–84.
[8] Arcos-Medina, G., Menéndez, J., & Vallejo, J. (2017). Comparative Study of Performance and Productivity of MVC and MVVM Design Patterns. SIIPRIN'2017.
[9] Lou, T. (2016). A Comparison of Android Native App Architecture – MVC, MVP and MVVM. Master's Thesis, Aalto University.