ISSN: 2583-9888
DOI: doi.org/10.51219/JAIMLD/naga-satya-praveen-kumar-yadati/177

URF PUBLISHERS
connect with research world

Journal of Artificial Intelligence, Machine Learning and Data Science

https://urfpublishers.com/journal/artificial-intelligence

Vol: 1 & Iss: 3

Research Article

Architecture Design (MVVM + Clean Architecture)

Naga Satya Praveen Kumar Yadati*

Citation: Yadati NSPK. Architecture Design (MVVM + Clean Architecture). *J Artif Intell Mach Learn & Data Sci 2023*, 1(3), 703-706. DOI: doi.org/10.51219/JAIMLD/naga-satya-praveen-kumar-yadati/177

Received: 02 September, 2023; **Accepted:** 18 September, 2023; **Published:** 20 September, 2023

***Corresponding author:** Naga Satya Praveen Kumar Yadati, USA, E-mail: praveenyadati@gmail.com

A B S T R A C T

The evolution of software architecture has seen the adoption of patterns that enhance maintainability, scalability, and testability. Two prominent paradigms in this evolution are the Model-View-ViewModel (MVVM) and Clean Architecture. This paper delves into the intricacies of combining these two architectures to build robust applications, particularly in the context of Android development. It aims to provide a comprehensive overview of both architectures, explore their synergy, and offer practical insights into their implementation.

Keywords: MVVM, Clean Architecture, Android Development, Software Architecture, Model-View-ViewModel, Separation of Concerns, Testability, Scalability, Maintainability, Data Binding, Presentation Logic, Business Logic, Data Layer, Use Cases, Entities, Interface Adapters, Frameworks and Drivers, Dependency Injection, Repository Pattern, Unit Testing, User Interface (UI), ViewModel, Modularity, Application Architecture, Code Reusability

1. Introduction

In the realm of software development, choosing the right architectural pattern is crucial for building applications that are not only functional but also maintainable and scalable. MVVM and Clean Architecture are two such patterns that have gained widespread adoption. This paper will explore how these two architectures can be combined to leverage their individual strengths, providing a detailed guide for developers aiming to implement this hybrid architecture.

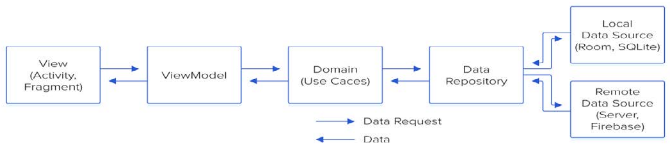
1.1. Background

- **MVVM**: Originating from Microsoft, MVVM is designed to provide a clear separation between the UI and business logic. This separation facilitates unit testing and enhances code maintainability.
- **Clean Architecture**: Proposed by Robert C. Martin, Clean Architecture emphasizes the separation of concerns, making the codebase easier to manage and scale. It divides the software into layers, with each layer having a specific responsibility.

1.2. Motivation

The motivation behind this paper is to bridge the gap between theory and practice, offering developers a structured approach to implementing a hybrid architecture that combines the best of MVVM and Clean Architecture.

2. Understanding MVVM


View (Activity, Fragment)
ViewModel
Domain (Use Caces)
Data Repository
Local Data Source (Room, SQLite)
Remote Data Source (Server, Firebase)
Data Request
Data

2.1. Definition and principles

MVVM (Model-View-ViewModel) is an architectural pattern that facilitates a separation of concerns within an application. It divides the application into three main components:

1. **Model**: Represents the data and business logic.
2. **View**: Represents the UI components.
3. **ViewModel**: Acts as a bridge between the Model and the View, handling the presentation logic.

2.2. Benefits of MVVM

- **Separation of Concerns**: MVVM clearly separates the UI from the business logic, making the code more modular and easier to manage.
- **Testability**: The ViewModel, which contains the presentation logic, can be easily unit tested without relying on the UI.
- **Data Binding**: MVVM leverages data binding to automatically synchronize the UI with the underlying data model.

2.3. Challenges of MVVM

- **Complexity**: For simpler applications, the overhead introduced by MVVM can be overkill.
- **Learning Curve**: Developers need to understand the intricacies of data binding and the interaction between the View and ViewModel.

2.4. Implementation details

In an MVVM architecture, the ViewModel is responsible for preparing the data for the View by interacting with the Model. The View binds to properties in the ViewModel, allowing it to automatically update in response to changes in the data. Here's a typical flow:

- **User Interaction**: The user interacts with the View.
- **ViewModel Update**: The ViewModel responds to the interaction by updating the data.
- **Model Interaction**: The ViewModel interacts with the Model to fetch or update data.
- **UI Update**: Data binding ensures that changes in the ViewModel are reflected in the View.

3. Understanding Clean Architecture

3.1. Definition and principles

Clean Architecture is an approach to software design that emphasizes the separation of concerns through the use of layers. Each layer has a specific responsibility and is isolated from the others. The primary layers in Clean Architecture are:

- **Entities**: Represent the core business logic and rules.
- **Use Cases**: Contain the application-specific business rules.
- **Interface Adapters**: Convert data from the format most convenient for the use cases and entities to the format most convenient for external agencies such as databases and the UI.
- **Frameworks and Drivers**: Contain the UI, database, web frameworks, etc.

3.2. Benefits of clean architecture

- **Maintainability**: The separation of concerns makes the codebase easier to maintain and modify.
- **Scalability**: The modular nature of Clean Architecture allows for easy scaling of applications.
- **Testability**: Each layer can be tested independently, improving the overall test coverage.

3.3. Challenges of clean architecture

- **Complexity**: Implementing Clean Architecture can introduce complexity, particularly in smaller projects.
- **Overhead**: The additional layers can lead to more boilerplate code and an increase in the initial setup time.

3.4. Implementation Details

In Clean Architecture, each layer is isolated from the others through the use of interfaces. This ensures that changes in one layer do not affect the others, promoting a more stable and maintainable codebase. Here's how the layers typically interact:

- **UI Layer**: Receives input from the user and passes it to the Use Case layer.
- **Use Case Layer**: Contains the business logic and interacts with the Entity layer.
- **Entity Layer**: Contains the core business rules and entities.
- **Data Layer**: Handles data persistence and retrieval, often interacting with external systems such as databases or APIs.

4. Combining MVVM and Clean Architecture

4.1. Synergy between MVVM and Clean Architecture

Combining MVVM and Clean Architecture allows developers to leverage the strengths of both patterns. MVVM provides a clear separation of concerns between the UI and the business logic, while Clean Architecture ensures a well-structured and maintainable codebase.

4.2. Architectural overview

In the combined architecture, the ViewModel from MVVM acts as the Interface Adapter in Clean Architecture. This allows the ViewModel to handle the presentation logic while interacting with the Use Case layer for business logic and the Entity layer for core business rules.

4.3. Benefits of the combined architecture

- **Enhanced Separation of Concerns**: The combination provides a clear separation between the UI, presentation logic, business logic, and data layers.
- **Improved Testability**: Each layer can be tested independently, leading to higher test coverage and more reliable code.
- **Scalability and Maintainability**: The modular nature of the combined architecture makes it easier to scale and maintain the application.

4.4. Implementation guidelines

- **ViewModel**: Acts as the Interface Adapter, handling the presentation logic and interacting with the Use Case layer.
- **Use Cases**: Contain the application-specific business logic and interact with the Entity layer.
- **Entities**: Represent the core business rules and entities.
- **Data Layer**: Handles data persistence and retrieval.

4.5. Practical example

- Consider an Android application for managing tasks. The combined architecture can be implemented as follows:
- **ViewModel**: Handles user input and updates the UI. It interacts with the Use Case layer to fetch or update tasks.
- **Use Cases**: Contain the business logic for managing tasks,

such as adding, deleting, and updating tasks.

- **Entities**: Represent the task objects and their associated business rules.
- **Data Layer**: Manages the persistence of tasks, interacting with a database or remote API.

5. Implementation Case Study

5.1. Project Overview

This section will provide a detailed case study of implementing the combined MVVM and Clean Architecture in an Android application. The project will involve building a task management application with features such as adding, updating, deleting, and viewing tasks.

5.2. Requirements

- **Add Task**: Users can add new tasks with details such as title, description, and due date.
- **Update Task**: Users can update existing tasks.
- **Delete Task**: Users can delete tasks.
- **View Tasks**: Users can view a list of tasks.

5.3. Architectural Design

The architectural design will follow the combined MVVM and Clean Architecture pattern, with each layer handling specific responsibilities.

5.4. Implementation Details

- **ViewModel**: Handles the presentation logic and interacts with the Use Case layer.
- **Use Cases**: Implement the business logic for adding, updating, deleting, and viewing tasks.
- **Entities**: Represent the task objects and their associated business rules.
- **Data Layer**: Manages data persistence, interacting with a local database or remote API.

5.5. Benefits and Drawbacks

Benefits

- **Separation of Concerns**: The clear separation between the UI, presentation logic, business logic, and data layers improves maintainability and scalability.
- **Testability**: Each layer can be tested independently, leading to higher test coverage and more reliable code.
- **Scalability**: The modular nature of the combined architecture allows for easy scaling of the application.

Drawbacks

- **Complexity**: Implementing the combined architecture can introduce complexity, particularly for smaller projects.
- **Overhead**: The additional layers can lead to more boilerplate code and an increase in the initial setup time.

6. Conclusion

Combining MVVM and Clean Architecture provides a robust framework for building maintainable, scalable, and testable applications. While the initial setup and complexity may be higher, the long-term benefits in terms of maintainability, scalability, and testability make it a worthwhile investment. This paper has provided a comprehensive overview of both architectures, explored their synergy, and offered practical insights into their implementation, serving as a valuable resource for developers looking to implement this hybrid architecture.

7. References

1. Martin RC. Clean architecture: A craftsman's guide to software structure and design. Prentice Hall 2017.
2. Microsoft Documentation. Model-View-ViewModel (MVVM) Pattern. 2012.
3. Android Developers. Guide to app architecture. Developers.
4. Uncle Bob. Clean code and clean architecture. Clean Code.
5. Fowler M. Patterns of enterprise application architecture. Addison-Wesley Professional 2002.
6. Allen G. Modern android development with jetpack compose. Packt Publishing 2020.
7. Gamma E, Helm R, Johnson R, Vlissides J. Design Patterns: Elements of reusable object-oriented software. Addison-Wesley 1994.
8. Brown K. Advanced android development: Bringing MVVM to android development. O'Reilly Media 2018.
9. Boyar Y, Powell A. Android architecture components: A comprehensive guide. Google I/O 2018.
10. Beck K. Test driven development: By example. Addison-Wesley 2002.
11. McCabe TJ. A complexity measure. IEEE Transactions on software engineering 1976;2: 308-320.
12. Bass L, Clements P, Kazman R. Software architecture in practice. Addison-Wesley Professional 2003.
13. Hevery M. A guide to writing testable code. Google testing blog 2008.
14. Koskimies K, Mikkonen T. Understanding software engineering. John Wiley & Sons 2005.
15. Johnson R, Hoeller J, Arendsen A, Harrop R, Risberg T. Professional Java Development with the Spring Framework. Wrox 2004.
16. Evans E. Domain-Driven design: Tackling complexity in the heart of software. Addison-Wesley 2003.
17. Burns C, Vignesh M. MVVM in android: Managing the view-model relationship. Manning Publications 2021.
18. Steele GL Jr. Common Lisp: The Language 2nd edition. Digital Press 1990.
19. Apple Inc. Swift Programming Language. Apple Books 2015.
20. Knuth DE. The Art of Computer Programming, Volumes 1-3 Boxed Set 3rd edition. Addison-Wesley Professional 1997.
21. Sutter H. Exceptional C++ Style: 40 New engineering puzzles, programming problems, and solutions. Addison-Wesley Professional 2004.
22. Bloch J. Effective Java 2nd edition. Addison-Wesley 2008.
23. Fowler M, Beck K. Refactoring: Improving the design of existing code. Addison-Wesley 1999.
24. Subramaniam V, Hunt A. Practices of an agile developer: Working in the real world. Pragmatic Bookshelf 2006.
25. Lewis J, Loftus W. Java Software Solutions 10th edition. Pearson 2019.
26. Sedgewick R, Wayne K. Algorithms 4th edition. Addison-Wesley Professional 2011.

27. Larman C. Applying UML and patterns: An introduction to object-oriented analysis and design and iterative development (3rd edn.). Prentice Hall 2004.
28. Pilone D, Pitman N. UML 2.0 in a Nutshell. O'Reilly Media 2005.
29. Meyer B. Object-Oriented software construction 2nd edition. Prentice Hall 1997.
30. Hunt A, Thomas D. The pragmatic programmer: Your journey to mastery. Addison-Wesley 1999.
31. Pressman RS. Software engineering: A practitioner's approach 7th edition. McGraw-Hill Education 2009.
32. Sommerville I. Software engineering 10th edition. Pearson 2015.
33. Hiltzik MA. Dealers of Lightning: Xerox PARC and the dawn of the computer age. HarperBusiness 1999.
34. Armstrong D. The quarks of object-oriented development. Springer 2006.
35. McConnell S. Code Complete 2nd editon. Microsoft Press 2004.
36. Nagy K. MVVM architecture for android developers: A practical guide. Leanpub 2021.
37. Misfeldt A, Hendrickson E, Kolawa A. Exploring test automation patterns. Wiley 2004.
38. Parnas DL. On the criteria to be used in decomposing systems into modules. Communications of the ACM 1972;15: 1053-1058.