3rd International Conference on Information Science and Technology Innovation (ICoSTEC)
July 27, 2024, Yogyakarta, Indonesia

Finding The Best Understandability for Android GUI Architectural Application Using Empirical Evaluation

Kurniadi[1], Yhudha Juwono[2], Umi Laili Yuhana[3]

Departement of Informatics, Institut Teknologi Sepuluh Nopember
Surabaya, Indonesia
[1]6025231002@student.its.ac.id
[2]6025231055@student.its.ac.id
[3]yuhana@if.ts.ac.id

***Abstract*—Software Understandability is a pivotal concept emphasizing the need for a system to be presented in a manner that engineers can effortlessly comprehend. When a system is inherently understandable, engineering operations become streamlined, promoting efficiency in the development process. The significance of code understanding cannot be overstated, as it directly impacts productivity, reduces errors, and fosters an environment conducive to innovation. This paper delves into the realm of software understandability, placing a specific focus on two mobile-based development architectures: MVVM and VIPER. These architectures are recognized for their ability to break down software components with enhanced abstractions, contributing to a more coherent development structure. The primary objective of this research is to quantitatively measure the best understandability between MVVM and VIPER through the application of measurement metrics extracted from both project using MetricReloaded and SonarQube. Upon analysis, the measurement results reveal that VIPER demonstrates superior understandability when compared to MVVM. This finding underscores the potential of VIPER to not only facilitate a more comprehensible development process but also to serve as a catalyst for increased innovation within the realm of software development.**

Keywords—MVVM, VIPER, Understandability, GUI Architecture

I. Introduction

In the domain of software development, a fundamental prerequisite for effective maintenance and evolution lies in the comprehension of software code. Without a deep understanding of the codebase, developers encounter significant challenges when it comes to debugging or enhancing software functionalities in a timely manner. The measurement of code understandability can serve as a valuable compass for developers, aiding the`m in crafting high-quality code [1]. Moreover, it can assist in estimating the effort required to modify code components. This introduction lays the groundwork for a closer exploration of the concept of 'software understandability' and its pivotal role in contemporary software engineering practices.

In the realm of mobile application development, especially in Android, there are popular architectures that are well-suited for use. MVVM (Model View ViewModel) and VIPER (View Interactor Presenter Entity and Router) are considered appropriate for applications across various scales—be it small, medium, or large. In addition to their excellent performance [2], [3], their efficient abstractions in architecture position them as preferable choices compared to MVC and MVP. [4].

MVVM is an architecture invented by Microsoft's architects Ken Cooper and Ted Peters, to aim simplicity of event driven programming interfaces. MVVM simplify two way communication between UI XML files with their C# View files. MVVM concept is a variation of on Martin Fowler's Presentation Model design pattern. VIPER is an architecture concept based on Uncle Bob's Clean Architecture. VIPER aim to have very loose code cohesion between each layer of abstraction. This architecture heavily influenced by Onion Architecture and used to promote SOLID principles of programming.

The main contribution of this research is to provide a deep understanding of the characteristics of understandability in the context of Android application development architectures, particularly in the comparison between the MVVM and VIPER architectures. By conducting a comprehensive evaluation of these two architectures, the research aims to identify key aspects influencing developers comprehension levels of code structure.

In this study, we are seeking the value of understandability from the MVVM and VIPER architectures. The aim is to obtain the best understandability scores that can be used to illustrate how developers can achieve a thorough understanding of the code. The content of this paper organized as: Section II showing related work to measure understandability. Section III discussing research methodology, Section IV explaining about our application with the experiment results, Section V we will give conclusions from our research and future work.

II. Related Works

Implementing design patterns in mobile application development can reduce complexity and improve maintainability [5]. Some research finds that MVVM is more maintainable and modifiable than MVP [6]. Implementing clean architecture has proven that the code is neater, more readable, and easier to maintain [7].

Understandability remains a pivotal quality attribute utilized to evaluate the clarity of object-oriented software [8]. It holds significance throughout various stages of the software development life cycle, as any misinterpretation during these phases can lead to the creation of a substandard product. Given that mobile applications, much like other software entities, undergo continuous maintenance and evolution, a thorough understanding of them is indispensable for ensuring their maintainability, reliability, quality, and reusability.

The measurement of software quality utilizes the ISO 91261 standard, which is divided into six characteristics, including

functionality, reliability, maintainability, usability, efficiency, and portability [9]. This method is applicable universally to all types of software, but there is no specific explanation addressing understandability.

One factor in understandability is the complexity of the program code. Previous research has proposed measuring complexity using metrics such as Line of Code (LOC), Halstead Complexity (HC), and Cyclomatic Complexity (CC)[10]. Recent studies have also introduced Cognitive Complexity, Code Readability, and CMI to gauge the level of understandability [11]. Cognitive Complexity, recognized in recent research by Marvin Boron [12], proves to be a superior alternative to other measures, especially when combined with McCabe's Cyclomatic Complexity [13]. Although challenging to measure, cognitive complexity must be assessed qualitatively [14].

The proposed metric for understandability aims to gauge object-oriented applications, providing a clearer framework for assessing the understandability index in an application [15]. However, it does not specifically address measurements for Android applications and process measures should be employed for enhanced code understandability [16].

Finally, Ahmad A Saifan [17] offered new equation of Android Understandability Index which we use in this paper to determine value of architecture Understandability Index. Little difference between us and their method is, we directly use average value of metric parameters instead of calculating it for each clases.

III. Proposed Method

In this section, we elucidate the overall design of the research. Fig. 1 illustrates the key steps involved in conducting the experiment. These steps align with those previously outlined in the literature review for assessing the understandability of MVVM and VIPER architecture. However, it's noteworthy that we employed distinct tools and incorporated all Android metrics presented in the literature to formulate a specific method for measuring the understandability of Android applications.

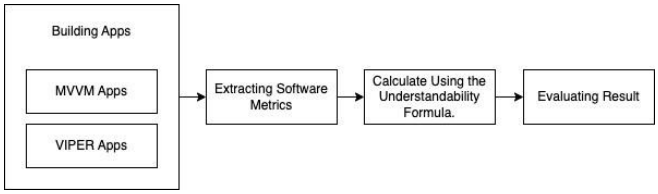

Fig. 1. Experiment diagram

A. Building Apps

Our methodology for evaluating the understandability of MVVM and VIPER involves the creation of two identical applications with distinct architectures. We developed two straightforward applications, both featuring the same functionality of loading data from local JSON files on the Main Page. When a user taps on an item, it navigates to the detail view. Additionally, both applications possess the capability to delete items from the list. A crucial principle of both MVVM and VIPER is to ensure that the view remains as simple as possible, avoiding the inclusion of heavy logic in it.

B. MVVM

The Model-View-ViewModel (MVVM) architecture is a software design pattern widely employed for developing user interfaces. It divides the application into three key components: Model, View, and ViewModel. The Model manages data and business logic, the View handles the user interface, and the ViewModel acts as a mediator between them. MVVM enhances code maintainability and scalability by1decoupling the user interface from the underlying logic. The architecture facilitates data binding, allowing seamless synchronization between the View and ViewModel. For a visual representation of the MVVM architecture, refer to the Fig. 2 and for real project structure of this architecture in android studio can be seen in Fig. 3

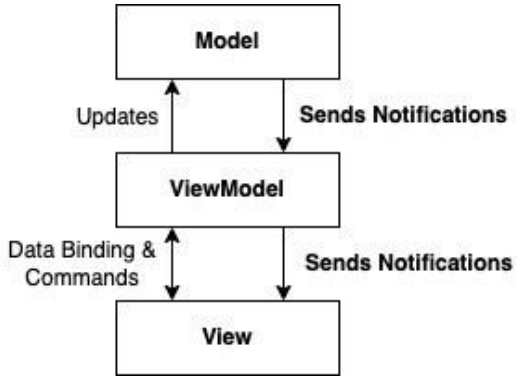

Fig. 2 MVVM architecture

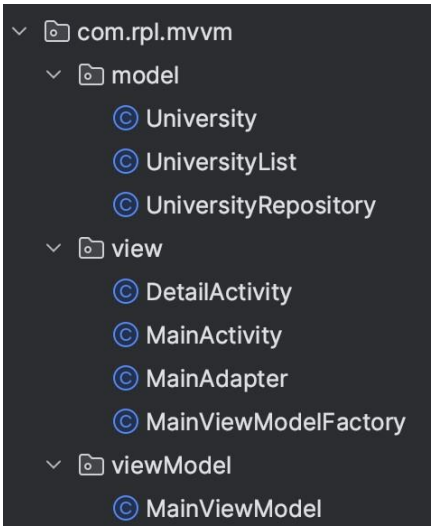

Fig. 3 MVVM directory Structure

B. VIPER

The VIPER architecture, standing for View-InteractorPresenter-Entity-Routing, is a robust design pattern commonly employed in software development, particularly for building scalable and maintainable iOS and Android applications. VIPER decomposes the application into five core components: View, Interactor, Presenter, Entity, and Routing. The View is responsible for the user interface, the Interactor manages business logic and data, the Presenter orchestrates the communication between the View and Interactor, the Entity encapsulates data objects, and Routing handles navigation between modules.

VIPER's modular structure enhances code organization, making it easier to understand, test, and modify. The explicit separation of concerns in VIPER contributes to a more

systematic approach to building complex applications. For a visual depiction of the VIPER architecture refer to the Fig. 4. Overall project structure used in this experiment can be seen at Fig. 5. Because VIPER have more abstraction it will be common to see VIPER have more directory than MVVM.

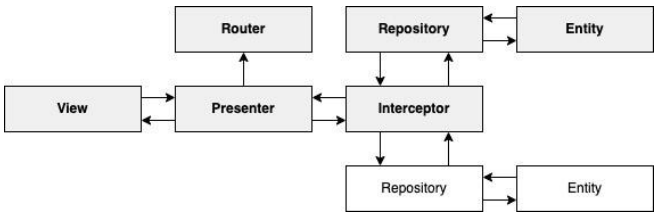

Fig 4. VIPER architecture

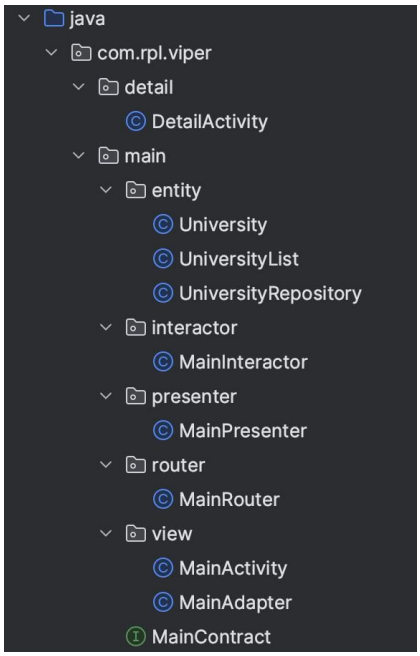

Fig 5. VIPER directory structure

Here are the Android applications based on the MVVM and VIPER architectures that we utilized as experiments in this research:

- MVVM Architecture https://github.com/kurniadi92/android-mvv-java
- VIPER Architecture https://github.com/kurniadi92/android-viper-java.

Previously our project created using Kotlin, but turns out MetricReloaded not working well with Kotlin so we need to rework on it using Java. For list of university, we get it from locally stored json file. No internet connection required to run the app. The reason why we have more than one page for this sample app is because we want to maximize VIPER implementation of router layer which described as part of the architecture who handle navigation.

The result of application can be see at Fig. 6. Both VIPER and MVVM have same look and same functionality. This app is a simple app that showing university list and then open university detail when you tap it. No login or authentication needed since we fetch list from locally stored json.

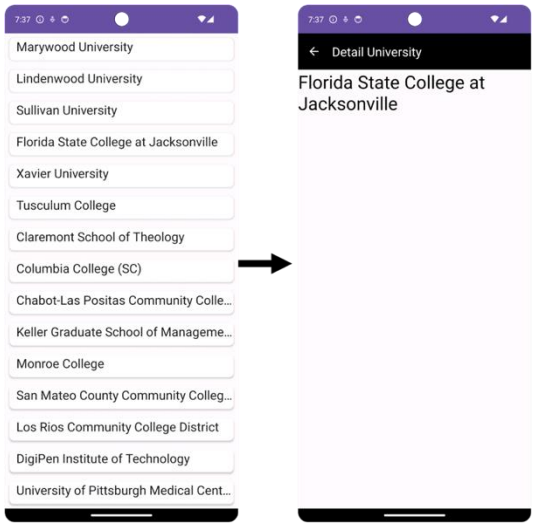

Fig. 6 Application interface in android emulator

C. Extract Software Metrics

In assessing the understandability metric, we employed two tools to extract the necessary parameters. By utilizing these two instruments, we were able to comprehensively measure the extent to which code can be understood by developers. This approach allows for a detailed and nuanced analysis of various facets of understandability in the process of evaluating code quality.

SonarQube, utilized as a static analysis tool, plays a pivotal role in the evaluation of software code quality by measuring both Cyclomatic Complexity and Cognitive Complexity. Through its robust capabilities, SonarQube provides insights into the structural intricacies of code, assessing its complexity and the cognitive load required for comprehension. The analysis generated by SonarQube aids developers in identifying potential areas for code improvement, enhancing overall code maintainability and readability. This comprehensive approach to static analysis contributes to a more informed and efficient software development process, aligning with best practices for ensuring code quality.

MetricReloaded is an invaluable tool for automating source code metrics in IntelliJ IDEA and IntelliJ Platform IDE. It provides essential metrics like Lines of Code (LOC) and file count for all supported languages, with additional tailored metrics for Java. This tool facilitates a comprehensive analysis of code complexity, maintainability, and other crucial factors, enhancing overall code quality understanding within the IntelliJ development environment. Subsequently, MetricReloaded is employed to analyze Halstead Complexity (HC), Halstead Effort (HE), and extract values for calculating the understandability index.

Subsequently, the results obtained from SonarQube and MetricReloaded were transferred to an Microsoft Excel file. For parameters that cannot be obtained from the mentioned tools, like INT, a manual process is undertaken. This involves calculating the number of interfaces implemented by each class. Following this, calculations were conducted to derive the understandability index results.

D. Calculate Using Understandability Formula

The dataset includes the understandability index, computed using the formula [17] specifically measures understandability index by relying solely on android application metrics.

IV. Results

The measurements for each parameter were systematically collected and meticulously processed utilizing Microsoft Excel. The resultant data is comprehensively presented in Table I and Table II and the definition of metrics referring to the research by A. Saifan, et al [17], facilitating a detailed examination of the metrics under consideration. This meticulous approach to measurement and analysis contributes to the accuracy and reliability of the obtained results, ensuring a thorough exploration of the specified parameters in the context of the study.

TABLE I
EXTRACTED SOFTWARE METRICS 1

Arch	INT	LCOM	CBO	SUB	NOC	WMC	CSOA	MPC	DIT
MVVM	3	1	5.56	0	0	3.11	218	7.89	3.11
VIPER	8	1.08	4.92	0	0	2.50	166.50	5.58	2.50

TABLE II
EXTRACTED SOFTWARE METRICS 2

Arch	OCAvg	CLOC	OCMax	LOC	JF	N	n	Inner
MVVM	1.04	0.56	1.14	21.67	0	41.56	23.78	0.22
VIPER	1	1	1	18.33	0	31.92	19.17	0.17

a. Arch = Architecture

Lastly, in Table III, we present the understandability metric utilized in this research. This metric is derived from the output generated by both tools employed in the study. The table not only includes the raw data obtained from these tools but also encompasses the calculated Understandability Index (UI). This index is formulated through a predefined formula, reflecting a comprehensive evaluation of the codebase's understandability based on the aggregated metrics.

TABLE III
UNDERSTANDABILITY METRICS

Architecture	HC	HE	CyC	CoC	UI
MVVM	11.87	3.526.48	28	3	-95.6
VIPER	8.75	2.229.62	28	2	-72.19

b. HC = Halstead Complexity
c. HE = Halstead Effort
d. CyC = Cyclomatic Complexity
e. CoC = Cognintive Complexity
f. UI = Understandability Index

A. Halstead Complexity

The Halstead Complexity metric is designed to correlate with the level of difficulty in understanding a class. In our experiment, we observed that MVVM exhibited a higher complexity compared to VIPER, despite the absence of heavy logic in both implementations. The complexity values were 11.87 for MVVM and 8.75 for VIPER. This suggests that, according to the Halstead Complexity metric, MVVM may present a greater challenge in terms of comprehensibility compared to VIPER, even in scenarios where intricate logic is minimal.

B. Halstead Effort

The Halstead Effort metric aims to reflect the level of effort required to understand a class. In our investigation, VIPER demonstrated a lower effort in understanding its class compared to MVVM. This observation aligns with our expectations, considering that Halstead Effort is typically expected to correlate with Halstead Complexity (HC). Therefore, the lower effort required for VIPER's class comprehension suggests that, according to the Halstead Effort metric, VIPER may present a more straightforward and less effort-intensive understanding compared to MVVM.

C. Cyclomatic Complexity

Cyclomatic complexity serves as a quantitative measure indicating the number of linearly independent paths through a program. A lower cyclomatic complexity is generally considered favorable. Interestingly, in our metrics, both MVVM and VIPER exhibit a tie in this aspect. Our conjecture is that this tie in results is attributed to the absence of intricate logic in either implementation, thus resulting in a comparable cyclomatic complexity. This observation aligns with the expectation that a lower cyclomatic complexity signifies a more straightforward and potentially more maintainable codebase.

D. Cognitive Complexity

Cognitive complexity determines the amount of human effort required to comprehend its internal logic, which results in a subjective measurement. It's means the larger the value means larger effort. For this metrics VIPER win over MVVM. The reason behind why VIPER have score 2 is probably because VIPER heavily compose their structure to very small pieces of object with specific responsibility for each layer. This means will be unexpected to have View which do a navigation process because this already handled by router. In other hand, MVVM still rely on their view to do navigation.

E. Understandability Index

The Understandability Index serves as a crucial indicator gauging the comprehensibility of a class or project, with negative values indicating lower understandability. In our experiment, VIPER demonstrated a superior Understandability Index with -72.19, compared to MVVM with -95.6. This composite index takes into account various factors, including encapsulation, coupling, abstraction, cohesion, polymorphism, complexity, and design. The notable difference in Understandability Index values suggests that VIPER exhibits a more understandable structure, attributed to favorable factors such as better encapsulation, reduced coupling, effective abstraction, improved cohesion, and a well-designed approach.

V. CONSLUSION

Prior to our experiment which builds upon very basic implementation of MVVM and VIPER, we can conclude that VIPER have better understandability in GUI Architectural design pattern. This is due to fact that VIPER have better result in four from our five metrics. VIPER with it's abstraction and layers give better understandability regarding of role and structure of the project. But this not means MVVM is very bad compared to VIPER. Our result shows there are just little different between MVVM and VIPER cognitive complexity. So even if VIPER have better understandability, MVVM is not bad too because they just have very little margin difference.

In this study, we focused exclusively on measuring one quality attribute, namely Understandability, within the MVVM and VIPER architectures. Future research endeavors are encouraged to explore the measurement of additional quality attributes to offer a more comprehensive evaluation of software architectures. Additionally, this study was conducted on a relatively straightforward codebase. It is imperative for subsequent research to extend its scope to more complex codebases, with the challenge of ensuring the equivalence of functionalities across applications. This is essential to

guarantee a fair and unbiased experimental environment, enabling a nuanced examination of how architectural choices impact various quality attributes in real-world, intricate software systems.

REFERENCES

[1] S. Scalabrino, G. Bavota, C. Vendome, M. Linares-Vasquez, D. Poshyvanyk, and R. Oliveto, "Automatically assessing code understandability: How far are we?," in *ASE 2017 - Proceedings of the 32nd IEEE/ACM International Conference on Automated Software Engineering*, 2017. doi: 10.1109/ASE.2017.8115654.

[2] B. Wisnuadhi, G. Munawar, and U. Wahyu, "Performance Comparison of Native Android Application on MVP and MVVM," 2020. doi: 10.2991/aer.k.201221.047.

[3] H. A. Epiloksa, D. S. Kusumo, and M. Adrian, "Effect Of MVVM Architecture Pattern on Android Based Application Performance," *JURNAL MEDIA INFORMATIKA BUDIDARMA*, vol. 6, no. 4, 2022, doi: 10.30865/mib.v6i4.4545.

[4] N. Akhtar and S. Ghafoor, "Analysis of Architectural Patterns for Android Development," *Conference: Analysis of Architectural Patterns for Android development-SDA*, vol. 1, no. 1, 2021.

[5] H. K. Jun and M. E. Rana, "Evaluating the Impact of Design Patterns on Software Maintainability: An Empirical Evaluation," in *2021 3rd International Sustainability and Resilience Conference: Climate Change*, Institute of Electrical and Electronics Engineers Inc., 2021, pp. 539–548. doi: 10.1109/IEEECONF53624.2021.9668025.

[6] A. Wilson, F. Wedyan, and S. Omari, "An Empirical Evaluation and Comparison of the Impact of MVVM and MVC GUI Driven Application Architectures on Maintainability and Testability," in *2022 International Conference on Intelligent Data Science Technologies and Applications, IDSTA 2022*, Institute of Electrical and Electronics Engineers Inc., 2022, pp. 101–108. doi: 10.1109/IDSTA55301.2022.9923083.

[7] A. Rahman Fajri and S. Rani, "Penerapan Design Pattern MVVM dan Clean Architecture pada Pengembangan Aplikasi Android (Studi Kasus: Aplikasi Agree Partner)," *Official Scientific Journals of Universitas Islam Indonesia*, 2022.

[8] M. Alenezi, "Software Architecture Quality Measurement Stability and Understandability," 2016. [Online]. Available: www.ijacsa.thesai.org

[9] Sholiq, R. A. Auda, A. P. Subriadi, A. Tjahyanto, and A. D. Wulandari, "Measuring software quality with usability, efficiency, and portability characteristics," in *IOP Conference Series: Earth and Environmental Science*, IOP Publishing Ltd, Apr. 2021. doi: 10.1088/17551315/704/1/012039.

[10] Y. Tashtoush, M. Al-Maolegi, and B. Arkok, "The Correlation among Software Complexity Metrics with Case Study," 2014.

[11] M. Saif Himayat and J. Ahmad, "Software Understandability using Software Metrics: An Exhaustive Review," *International Journal of Engineering and Management Research*, vol. 13, no. 2, 2023, doi: 10.31033/ijemr.13.2.31.

[12] M. M. Barón, M. Wyrich, and S. Wagner, "An empirical validation of cognitive complexity as a measure of source code understandability," in *International Symposium on Empirical Software Engineering and Measurement*, IEEE Computer Society, Oct. 2020. doi: 10.1145/3382494.3410636.

[13] G. Rotoloni, "Analysis of the performance of Cognitive Complexity and evaluation of alternative metrics and solutions using real understandability data. Analysis of the performance of Cognitive Complexity and evaluation of alternative metrics and solutions using real understandability data." [Online]. Available: https://www.researchgate.net/publication/366569694

[14] D. R. Wijendra and K. P. Hewagamage, "Application of the Refactoring to the Understandability and the Cognitive Complexity of a Software," in *2022 IEEE 7th International conference for Convergence in Technology, I2CT 2022*, Institute of Electrical and Electronics Engineers Inc., 2022. doi: 10.1109/I2CT54291.2022.9824082.

[15] S. Rajesh and A. Chandrasekar, "Metrics measurement model: To measure the object oriented design metrics," in *ICoAC 2015 - 7th International Conference on Advanced Computing*, Institute of Electrical and Electronics Engineers Inc., Sep. 2016. doi: 10.1109/ICoAC.2015.7562793.

[16] L. Lavazza, S. Morasca, and M. Gatto, "An empirical study on software understandability and its dependence on code characteristics," *Empir Softw Eng*, vol. 28, no. 6, Nov. 2023, doi: 10.1007/s10664-023-10396-7.

[17] A. A. Saifan, H. Alsghaier, and K. Alkhateeb, "Evaluating the Understandability of Android Applications," *International Journal of Software Innovation*, vol. 6, no. 1, pp. 44–57, Jan. 2018, doi: 10.4018/IJSI.2018010104.