Journal of Theoretical and Applied Information Technology
15th August 2019. Vol.97. No 15

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

REVIEW OF IOS ARCHITECTURAL PATTERN FOR TESTABILITY, MODIFIABILITY, AND PERFORMANCE QUALITY

[1]FAUZI SHOLICHIN, [2]MOHD ADHAM BIN ISA, [3]SHAHLIZA ABD HALIM, [4]MUHAMMAD FIRDAUS BIN HARUN

[1,2,3]Software Engineering Research Group, Faculty of Engineering, Universiti Teknologi Malaysia, Malaysia

[4]Software Construction Research Group, RWTH Aachen University, Germany.

E-mail: [1]sholichin.fauzi@graduate.utm.my, [2]mohdadham@utm.my, [3]shahliza@utm.my
[4]firdaus.harun@swc.rwth-aachen.de

ABSTRACT

In the mobile development especially in iOS, a correct selection of architecture patterns is crucial. Many architectural patterns used by developers such as Model View Controller (MVC), Model View Presenter (MVP), Model View ViewModel (MVVM), and View Interactor Presenter Entity Router (VIPER) have promised stability of the product. Nowadays, most developers tend to use MVC architectural pattern as this pattern is easy to use and separate the logic between model, view and controller. However, this architecture has common problems which are hard to test and manage the code because all the codes for business application are placed in controller components. Therefore, this paper reviewed some of the existing architectural patterns qualities specifically in testability, modifiability and performance quality in order to investigate the mentioned problems. By using Contact mobile apps as a case study, the results show the MVVM architecture is good for testability, modifiability (cohesion level procedural), and performance (memory consumption). In addition, VIPER is the best in modifiability (coupling level data and coupling level message) and performance of CPU.

Keywords: *Architecture Pattern; Design Pattern; Software Architecture; Quality attributes; Software Professionals*

1. INTRODUCTION

Mobile phones are increasingly expanding universality among clients. Ios and Android devices are generally accessible, while the market rivalry between various gadgets has been severe. Based on a survey in free web intelligence report in August 2018, iOS made steady headway in the US, France, Canada, Japan, and South Africa [1]. Hence, the market for iOS is broad and has the opportunity for a startup to develop their business to consumers through the iOS application.

To date, most of the company has developed a product in the iOS and Android platform. Usually, the development takes 18 weeks to produce a first version or minimum viable product. The result based on 100 mobile architects realizes how long they expected to develop it [2]. The success of the development of the app not only depending on the programmer skills but also from the selected architectural patterns. Thus, the selection of the architecture pattern is crucial in a software design phase.

In recent years, most of the iOS apps are designed using MVC, that splits the application into three layers: model, view, and controller, and where the default role of the controller is to link the two other layers [3][4]. The controller layer in MVC for iOS offer explicit responsibilities other than just a connection between model and view layer. All these added responsibilities make the controllers massive and complicated and also thighly coupled between view and controller layer. The problem is getting even more serious in a medium and large scale project where this scenario will have a huge impact to the software quality [3].

Previous work, are looking for the relation of architecture that is actually giving an impact to a software quality indirectly. Other works stated that

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

MVVM / MVP is better than MVC based on a variety of qualities on Android OS environment settings specifically for modifiability, testability, and performance [5].

Differences of this research compared to other similar work in this area of research is twofold: first, we are focusing on iOS platform, supported by a third party library such as Alamofire and SwiftyJson. Second, we investigate the VIPER architecture where it is claimed to be the best of architecture in testability and distribution [4]. Furthermore, there is no observational data report similar to what have been done in this paper to support VIPER as a new architecture with qualities such as modifiability, performance, maintainability [6].

The primary challenge to the problem is to choose a suitable architecture pattern for product development that promises a good software quality. In addition, a various software company faces a problem when they have a bottleneck; even a few has to rewrite code even from scratch. Rewrite a code from scratch contribute a significant impact to the effort for the company and developers. Thus, to avoid this problem, a process to choose the right architecture pattern is crucial.

The study was conducted in the area of software architecture on iOS native development by the problems arising from an industrial based problem. Problems such as lack of MVC for large-scale project consequently influence the quality of modifiability, testability and performance of an application. In this sense, MVC architecture is becoming irrelevant when a Massive View Controller syndrome appears in a controller. As a consequences, the product is difficult to test in display controller, which will influence testability and modifiability quality. In this study, the problem of choosing the right iOS architecture for a large-scale project is our main focus, where the suitable architecture will improve the modifiability in an iOS project which already complex, thus improving the performance of an iOS application.

2. CURRENT WORKS

In this section, we investigate a current works in architectural patterns for mobile apps development on IOS platform. In attempting to understand the landscape of the leading research area, literature has previously focused on architecture pattern in mobile development, namely: model view controller (MVC), model view ViewModel (MVVM), model view presenter (MVP), and view Interactor presenter entity router (VIPER). This section also presents prior works related with mobile architecture pattern and discusses any possible research gaps.

2.1 Software Architecture Pattern

Architectural patterns have always helped to build a testable, manageable and optimized software performance. It usually helps modularize the software so that each component is separated and handles a single responsibility. They also significantly improve the usability of a code, which performs a critical function in connecting the coding software. The software construction process also accelerates dramatically with the already proven design paradigms and mobile developers get more benefits in the development process with following the architectural patterns. Since the mobile application is getting bigger, hence mobile developer necessitate considering the design patterns before they go into the development application process. Several studies have shown that 50% to 70% of the total lifecycle cost for a software system is spent on evolving the system [7]. Software pattern architecture has an essential role in the process of changing, refactoring, and rewriting function or feature in the software development process.

A. Model View Controller

The MVC design pattern considers there to be three types of objects: model, view, and controller. Model objects encapsulate the data specific to an application and define the logic and computation that manipulate and process that data [8]. A view object is an object in an application purpose of display data from the model object that end user can see. Controller objects purposely to ast as an intermediary between one or more objects in view and model. Many objects in these applications tend to be more reusable, and their interfaces tend to be better defined [8].

MVC is a default architecture pattern in iOS native development. UML class diagram is illustrated in Figure 1 which gives view linked to the Controller class. View in iOS development could use Xib, Storyboard, or swift class to generated view. Xib and Storyboard not swift class, but user interface representation of an iOS application. In iOS, when using Xib class usually must declare a swift class as Controller with extending UIViewController class from UIKit native library in iOS. Model is typically plain Swift object class with responsibility for business logic. For example, regain data (text, photo, and etc.) from consumed RESTful API.

Journal of Theoretical and Applied Information Technology
15th August 2019. Vol.97. No 15

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

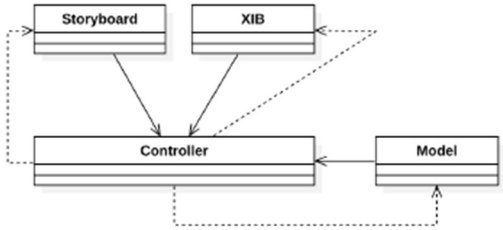

Figure 1. MVC IOS Implementation UML Class Diagram

A piece of an iOS app's architecture is necessary to view controller communication between controllers with another controller. For example, in Figure 2, which shows controller, connected with another controller, which every single controller connected with Model and Network Singleton for consumed RESTful API. View controller communication acquaints a strict sequel between view controllers. Architecture increased to the already complex execution flow of MVC.

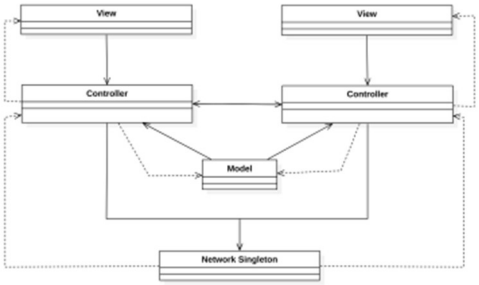

Figure 2. MVC IOS Implementation UML Class Diagram

2.1.2 Model View Presenter

The MVP pattern was invented in the 90s as a current C++ initiative from Apple, IBM, and HP as an alternative to the MVC pattern [9]. MVP has six components, which are a view, model, commands, selections, presenter, and interactor. Model element indicates a data in the application and share a similar concept in MVC. Sections component specified the part of the data to operate. Commands component deliver actions that could be execute. Presenter component that aims organize and coordinate all the same as the controller in MVC. Interactor component demonstrates the events that will be triggered by the user action. View component represented a view, which is similar in MVC. Although in original MVP, six components are defined, implementation in real-world development used only three components, which are Model, View, and Presenter. Hence, commands, selections, and interactor included in Presenter.

Figure 3 shows a UML class diagram implemented in the Swift project. In this version using Xib and View class as a view with import UIViewController from UIKit. Presenter decelerated in View class, which is view class could access function in a presenter. View Class extends a protocol class as an interface, so that could trigger callback action from presenter class. Callback action from presenter based on result data computation, response RESTful API, and logic depend on a case study. For example for a login case study, after the user clicked the login button, a view will call a function in the presenter. Then, the Presenter will throw the set of data to the model for the validation process. The result of process validation has typically success and failure response that result throws with a delegate to view for make a result view feedback.

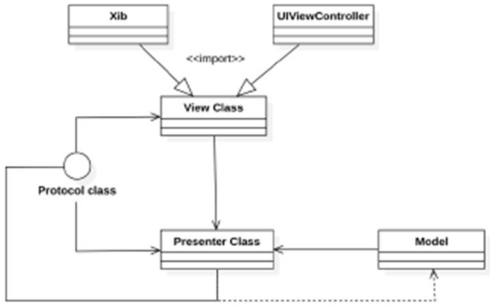

Figure 3. MVP IOS Implementation UML Class Diagram

2.1.3 Model View ViewModel

The MVVM is an architectural pattern most commonly used in Windows applications. The Architecture pattern was formulated by Ken Cooper and Ted Peters while working at Microsoft. John Gossman first announced this pattern in his blog in 2005 [10]. Model is the data or business logic, completely UI independent that stores the state and does the processing of the problem domain. The view consists of the visual elements display data that binding from the view model. View model Mediate the interaction between the Model and View. It passed data also manage the view's logic and behavior.

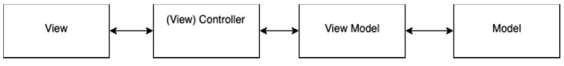

Figure 4. MVVM IOS Implementation Diagram

In MVVM, the Swift class extends with UIViewController that considered as View. View only knows how to present the data they are pass from View Model Class. In View Model, the core function is managing and preparing and the data for a View. It also handles communication data from local data and the rest of the application. After view was created, then View class calls the rest function by view model.

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

View model will execute logic and data that needed showing to view class.

2.1.4 View Interactor Presenter Entity Router

Viper was created at Mutual Mobile, an agency in Austin, Texas in 2014. After using and publishing the architecture, it became more common. The word VIPER is an acronym for View, Interactor, Presenter, Entity, and Routing.

View component like in MV(X) architecture to display and user input back to the presenter. Interactor focus on business logic as by use case. Presenter component like MVP architecture, but has other function different. Presenter received a data from interactor that displays to preparing content with view logic; by requesting new data from the interactor, that give reacting to user inputs. The interactor like Model component uses entity as a primary model objects. Routing as navigation logic for describing which screens are shown in which order.

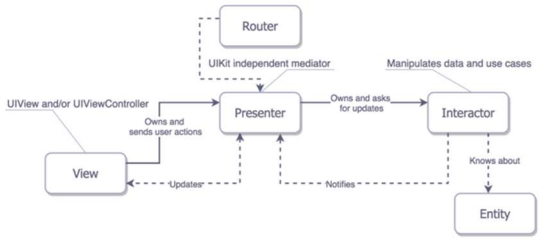

Figure 5. VIPER IOS Implementation Diagram

2.2 Comparative Prior Works in Software Architecture

Table 1 shows comparison prior works in software architecture pattern in criteria research type, architecture, and evaluation of research. The view of the comparison means to exist the advantages and disadvantages of prior works research with different architecture pattern. The view of the comparison means to exist the advantages and disadvantages of prior works research with different architecture pattern. The research study offers a development using MVP Architecture enhance with clean architecture, dependency injection, and reactive programming [11]. Clean architecture and dependency injections provide likelihood to make application easy to test as well as easy to add new features, which are features have to follow the dependency rules.

Table 1: The Comparison of prior works in Mobile Software Architecture Pattern

Authors	Title of Paper	Architecture	Advantages	Disadvantages
Duy, T. B. (2017)	Reactive Programming and Clean Architecture in Android Development	MVP	Explain detail of third party library (RxJava, Retrofit) which was implemented in architecture pattern design.	Not explain detail qualities impact of implementation software architecture.
Lou, T. (2016)	A comparison of Android Native App Architecture MVC, MVP and MVVM	MVC MVP MVVM	Architecture evaluation methods using ATAM (Architecture Trade-off Analysis Method)	Case study feature for comparative testing not specifically defined.
Felix Javier Acero Salazar, M. B. (2015)	Tailoring Software Architecture Concepts and Process for Mobile Application Development	MVVM VIPER	Present an advance level process and several drafts that aim to clue developers in the fit creation architectures for their apps	Not explain of step evaluate software architecture.
Giedrimas, V. and S. Omanovič (2015)	The Impact of Mobile Architectures on Component-based Software Engineering	MVC MVVM	Description differences between android and iOS component in architecture pattern	Not explain better option architecture for development process.
Syromiatnikov, A. and D. Weyns (2014)	A Journey Through the Land of Model-View-* Design Patterns	MVC MVP MVVM	Explain pros and cons of MVC, MVP, and MVVM	Test case just using MVP pattern, which is not comparing every single architecture

Further research studies, offer metrics framework for evaluating architecture considers multiple quality attributes such as performance, modifiability, security, and reliability [6]. Furthermore, the metrics framework made with some modifications based on ATAM method. ATAM has six phases are a collect scenario, collect requirement, describe architecture view, attribute-specific analysis, identify sensitives, identify tradeoffs. Thus, the architecture displayed in the research is modified from an android native application.

Implementation component in Android and iOS have differences. Example, component control in Android have two types are activity and fragment, while in iOS have one that ViewController. A paper discusses significant changes in the component concept influenced by the mobile platform [10]. The paper offers, differences of the software for mobile devices could be handled as the modern challenges in software engineering. Although, the platform of mobile changes in market side the component-based paradigm adapts from under evolution.

One of research that present a high-level process and several concepts that aim to the creation of suitable architectures mobile [17]. This research offers three processes to guide developers in creating an architecture for their apps. First, based on

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

important architectural processes. Second, to mapping the central concept and knowledge software architecture in real word mobile development. Third, to recommend for a different view of mobile software architectures, which is process and methodology are favored over predefined solutions. Thus, the research aims to extend the perspective of mobile software architectures. Moreover, in real-world mobile development well known that MV(X) architecture pattern. Research explain the journey of through in MVC, MVP, and MVVM that analyze pros and cons of every single architecture pattern [12]. Although, the case study of comparing architectures show with MVP, the decision result detail and inline to help practitioners to make a better
selection for choosing architecture.

B. Discussion

A software architecture have a major contribution towards a successful software project. At the beginning of development, it may take some time to design a good architecture. However, the effort will be paid off after the architecture has been developed with consideration of suitable qualities. One of the advantages is the ability to refactor a poor quality code caused by a bad architecture.

One of attribute comparison is an evaluation, that means disadvantages from uncovered in their research. The research not explains detail in terms of qualities impact of implementation using MVP [11]. The result of the research offer conclusion implemented with a clean architecture based on MVP is the code could easier for test and add a new feature. Indirectly state that with implementation architecture correctly could increase a quality of testability and modifiability of software.

Quality is part of essential things that practitioners are selecting the software architecture. The research proposes comparison software architecture based on testability, modifiability, and performance quality in android platform [5]. However, case study feature for comparative testing not explicitly defined, only exist the to-do apps in an appendix. Moreover, metrics formula for the subject is not included in this research. The advantage of this research is using ATAM method which one has an excellent flow to relate structurally. ATAM has six phases, which every single phase described based on prior work and requirement from a goal of phase.

3. METHODOLOGY

In general, the methodology describes research groundwork concerning definitely which research elements are required and what particular terms are applied to realize the whole research study.

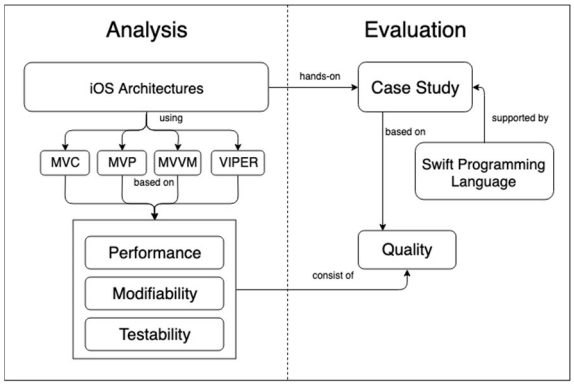

Figure 6. Methodology

3.1 Testability

3.1.1 Size of test cases

Le Taraon and Baudry proposed that the software testability influence by three parameter namely global test effort, controllability, and observability [13]. Moreover, test sets size is one of a part in global test effort to realize the aim of test. Unit test (XCode) feature used to calculate test cases size. Thus, **the architecture with less test cases is better.**

3.2 Modifiability

3.2.1 Cohesion level

The former is aimed to decide the cohesion component level by investigating tasks and data items inside the component cohesion [13][14]. The cohesion component strength is decided by analyzing the dependencies number inside this component as shown in Equation (1).

$$S_{Cohesion} = \frac{\sum_{i=1}^{m} A_i}{m}, A_i = \frac{\mu_i}{{n_i}^2} \qquad (1)$$

Ai is the strength of dependency within the i^{th} component. μ_i refers to the number of dependencies within the i^{th} component. n_i is the number of tasks within the component, and m is the number of components within the generated software architecture [13] [14].

Xulin Z, Fouts K, and Ying Z defines cohesion levels in the order from the worst namely low cohesion to high cohesion as to the top. Table 2 determines the data characteristics items and tasks in components for assessing the cohesion of components.

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

Table 2: Cohesion levels and their characteristics

Category	Cohesion level	Characteristics
Moderate levels	Procedural	Task within component is connected by control connectors.

3.2.2 Coupling level

The goal of components coupling is to assess the interdependencies strength between components with broken down into 7 distinct level [13] [14]. The coupling strength is measured based on the average of coupling strength among components. Researcher claimed the coupling between two components is assessed by examining the number of dependencies between the two components using Equation (2) [13] [14].

$$S_{Coupling} = \frac{\sum_{i,j=1}^{m} E_{i,j}}{m(m-1)/2}, E_{i,j} = \begin{cases} 0 & i = j \\ \frac{e_{i,j}}{2n_i n_j} & i \neq j \end{cases} \quad (2)$$

$\boldsymbol{E_{i,j}}$ is the inter-dependency between the i[th] component and the j[th] component. $\boldsymbol{E_{i,j}}$ is the total number of dependencies between the two components. Number of task are represented by two components $\boldsymbol{n_i}$ **and** $\boldsymbol{n_j}$. **m** is the total number of components in the generated software architecture [13] [14].

Xulin Z, Fouts K, and Ying Z defines coupling levels in the order from the worst namely high coupling to low coupling as to the top. Table 3 determines the data characteristics items and tasks in components for assessing the components coupling.

Table 3: Coupling level and their characteristic

Low levels	Data	Primitive data or arrays of primitive data are passed among components.
	Message	Components communicate through standardized interface

3.2.3 Measuring amount of functionality

Following functional size measurement, we re-mapped functionalities into the following five functional factors [13] [14]:

Table 4: Five functional factors

Functional Factors	Description
Internal logic files	which hold data items used within a component
External interface files	which contain external data received from the operational environment.
External inputs	which refer to input pins of tasks that hold external data
External outputs	which correspond to output pins of tasks that return data to the operational environment
External inquiries	which are tasks that capture data access actions.

For each functional factor, the FSM approach defines three levels of complexity and specifies a weight for the functional factor at each complexity level [13] [14]. Complexity levels and corresponding weights for the five functional factors shown in a table below:

Table 5: Weights of functional factors in FSM

Functional Factors	Low
Internal logic files	7
External interface files	5
External inputs	3
External outputs	4
External inquiries	3

3.3 Performance

3.3.1 Consumed memory

The metrics for consumed memory based on average of allocation memory for application per millisecond. To calculate of consumed memory using XCode Instrument. Thus, **the architecture with less consumed memory is better**

3.3.2 Consumed central processing unit

The metrics for consumed CPU based on average of allocation CPU for application per millisecond. To calculate of consumed CPU using XCode Instrument. Thus, **the architecture with less consumed CPU is better**

3.4 Case Study

The case study used for this study is Contact App. This case study is used to prove the usefulness

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

of the research outcomes concerning modeling and analysis of architecture impact to software quality in iOS native.

3.4.1 Contact App

Contact App is an application for looking for a contact phone, each of smartphone default has a contact application included. However, another developer has extended the contact phone app for custom features included in this app. Moreover, another application messenger and social media have a feature contact app. Thus, contact app features depend on the aim of application and user.

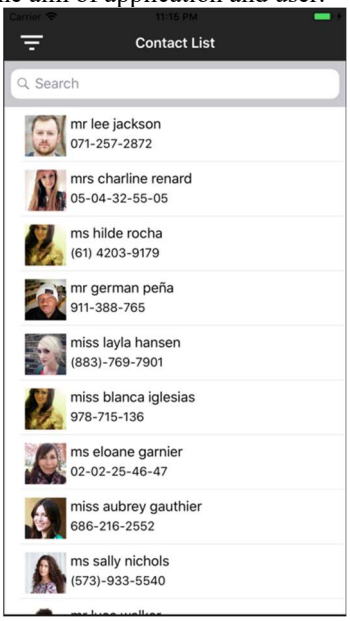

Figure 7. Contact App

In this case study, contact data based on public API randomuser.me. Randomuser.me is a free open-source API for generating random user data people created by Keith. A and Arron J.H [38]. The API will provide a default formatted in JSON. However, the API could custom formats such as CSV, YAML, and XML. Moreover, the API always updated features or information every year. Thus, a current number of data users generated are more than ten billion users per year.

3.4.2 Justification and Classification of the Case Studies

Table 6 shows the case studies and relevant criteria. The justification of the case studies choice is based on research contributions. Each contribution is proved by the selected case study in order to show the applicability of the findings.

Table 6: The justifications of the case studies

Criteria	Contact App
Feature to be Tested	- list contact - search contact - detail of contact
Data source	https://randomuser.me
Architecture Pattern	VIPER, MVP, MVVM, MVC
Quality to be measured	Modifiability, Performance, and Testability

Based on Table 6, the feature to be tested for every app have different feature and data source. Data source get from public data that could consume with JSON formatted. The case studies developed by Swift programing language based on every Architectures namely MVC, MVP, MVVM, VIPER.

A. Integrated Development Environment (IDE)

XCode

XCode is IDE for macOS which containing SDK for native iOS development. XCode supports C, C++, Objective-C, Objective-C++, Java, AppleScript, Python, Ruby, and Swift. Swift is a most common language to develop in iOS, Mac, and Apple TV. XCode has a feature to shows the debug navigator CPU, memory, disk, and network. This feature covered application when running on simulator or device. Figure 8 and Figure shows the graph percentage used and time when an application in a running test. The debug navigator will be turn off when application debugs stops.

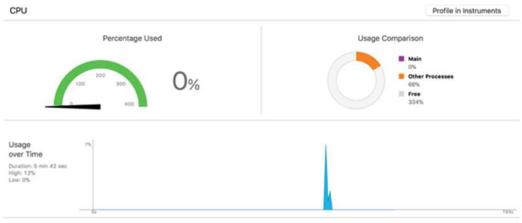

Figure 8. Debug CPU Navigator

Figure 9. Debug Memory Navigator

Journal of Theoretical and Applied Information Technology
15th August 2019. Vol.97. No 15

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

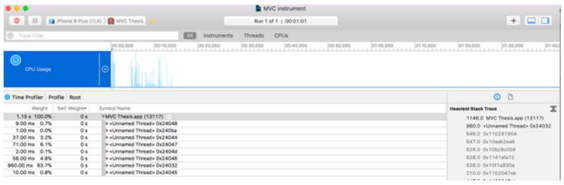

Figure 10. Debug Memory Navigator

Feature debug navigation has a specific app to record when application testing. The name is an instrument that has many features related to an analytic application. Figure 11 shows a menu of instrument namely activity monitor, leaks, time profiler and others.

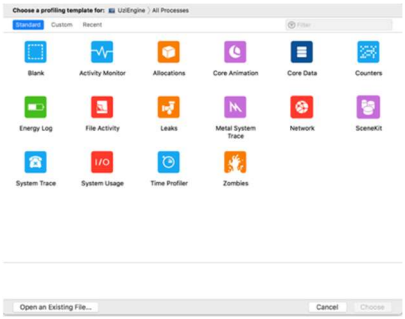

Figure 11. Instrument Menu Features

B. Device Testing

Testing application could use simulator and device. XCode provided simulator from iPhone 4 until iPhone X included in XCode after installed. However, build using real device have to used bundle ID, which is could download in developer.apple.com. The Certificate of apple developer also used to upload application to app store.

Device test used in this research is iPhone 5. Therefore, iPhone 5 has the lowest iPhone with supported iOS 10 with specification namely display phone 4.0 inch, processor 1.3 GHz, and RAM 1 Gb.

4. RESULT

4.1 Case Study

The result of the experiment will be explained based on the quality metrics, which are testability, modifiability and performance.

4.1.1 MVC

Model View Controller (MVC) is an architecture pattern with three components. Figure 12 shows the concept flow of MVC in a business application. Documentation MVC provided by Apple, so if follow the instruction of documentation

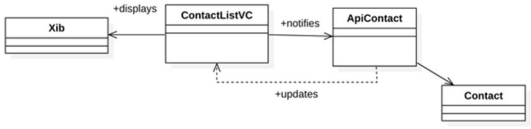

Figure 12. MVC Class Diagram

4.1.2 MVP

Model View Presenter (MVP) has three component View, Model, and Presenter. Implementation MVP in the iOS project needs to know about the delegation pattern. Delegation concept in swift bridged by protocol class. Each presenter has a protocol to communicate with View and protocol for declare method in a presenter.

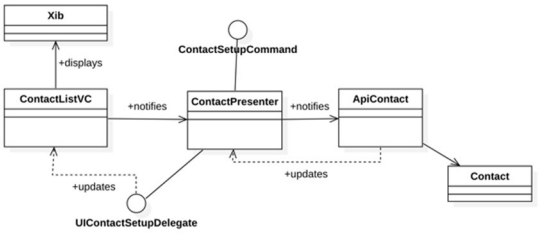

Figure 13. MVP Class Diagram

```
13 public protocol UIContactSetupDelegate {
14    func didLoadFinish(contact : [Contact])
15    func clearTableView()
16    func showError(m : String)
17 }
18
19 protocol ContactSetupCommand {
20   func loadContact()
21 }
22
23 class ContactListPresenter: ContactSetupCommand {
24     private var delegate: UIContactSetupDelegate
25     var lists: [Contact] = []
26
27     public init(with delegate: UIContactSetupDelegate) {
28         self.delegate = delegate
29     }
```

Figure 14. Class Abstract for Binding

```
208 extension ContactListViewController : UIContactSetupDelegate {
209     func showError(m: String) {
210         //show error
211     }
212
213     func clearTableView() {
214         self.rooms.removeAll()
215         self.roomsfilter.removeAll()
216     }
217
218     func didLoadFinish(contact: [Contact]) {
219         self.rooms = contact
220         self.roomsfilter = contact
221         self.tableView.reloadData()
222     }
223 }
```

Figure 15. Class Abstract for Binding

Figure 14 and 15 is example in case study Contact. Inside of UIContactSetupDelegate protocol as method callback that which will be forwarded from the presenter to the view when protocol activated.

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

Besides, ContactListPresenter extend ContactSetupCommand which is automatically declaration method and variable in ContactSetupCommand will implement inside ContactListPresenter class.

4.1.3 MVVM

Implementation MVVM in the iOS project needs an abstract class to binding data in ViewModel component. In Figure 17 shows class Observable which has a function of binding data. Obervable class implemented concept of observer pattern that can updated automatically when value change or update. Inside the class have three functions and two setup values for declaration data that want to bind. Syntax "T" mean value could set with anything type data start from a string, integer, long, double and so on until data object model. For example, in this case, Contact app has a Contact model class as data object shown in Figure 16.

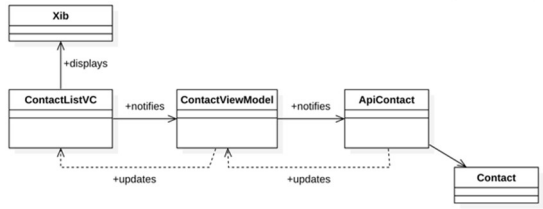

Figure 16. MVVMM Class Diagram

```
class Observable<T> {
    var value: T {
        didSet {
            DispatchQueue.main.async {
                self.valueChanged?(self.value)
            }
        }
    }

    private var valueChanged: ((T) -> Void)?

    init(value: T) {
        self.value = value
    }

    func addObserver(enable: Bool = true, _ onChange: ((T) -> Void)?) {
        valueChanged = onChange
        if enable {
            onChange?(value)
        }
    }

    func removeObserver() {
        valueChanged = nil
    }

}
```

Figure 17. Class Abstract for Binding

In Figures 17 shows the usage of class observer binding variable list, isError, and isLoading. Usually declaration variable in class like roomsFilter which is after equals that value of data.

```
class ContactViewModel {
    private var dataService: ApiContact?

    // MARK: - Properties List
    public var roomsfilter : [Contact] = []
    public var list = Observable<[Contact]>(value: [])

    // MARK: - Properties
    public var isError = Observable<String>(value: "")
    public var isLoading = Observable<Bool>(value: false)

    // MARK: - Constructor
    init(dataService: ApiContact) {
        self.dataService = dataService
    }
```

Figure 18. Class Abstract for Binding

```
    func initBinding(){
        command.list.addObserver(enable: true) { (contactList) in
            self.tableView.reloadData()
        }
        command.isLoading.addObserver { (isLoading) in
            //do loading view
        }
        command.isError.addObserver { (isError) in
            //show error
        }
        command.getContact()
    }
```

Figure 19. Class Abstract for Binding

In Figure 19 shows initiation data binding used in class extend UIViewController which is at MVVM that class as View component. For example, ContactListVC class in contact case study

4.1.4 VIPER

VIPER has five components and five a protocol classes in one flow of case. Implementation VIPER in the iOS project needs to know about the delegation pattern. Delegation concept in swift bridged by protocol class same like MVP. In Figure 21 shows an all of the protocol class in VIPER. Presenter and Interactor have a protocol to communicate with View and protocol for declare method in a presenter. Router component used protocol class for declare method inside class.

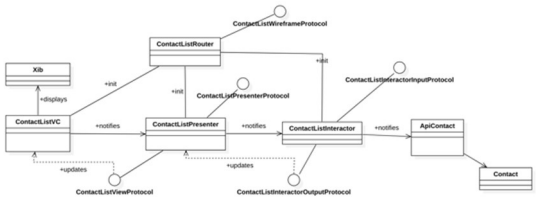

Figure 20. VIPER Class Diagram

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

```
//MARK: Wireframe -
protocol ContactListWireframeProtocol: class {

}
//MARK: Presenter -
protocol ContactListPresenterProtocol: class {

    var interactor: ContactListInteractorInputProtocol? { get set }
    func getContact()
    func getContactGender(gender : String)
}

//MARK: Interactor -
protocol ContactListInteractorOutputProtocol: class {

    /* Interactor -> Presenter */
    func showResponse(lists : [Contact])
}

protocol ContactListInteractorInputProtocol: class {

    var presenter: ContactListInteractorOutputProtocol?  { get set }

    /* Presenter -> Interactor */
    func getContact()
    func getContactGender(gender : String)
}

//MARK: View -
protocol ContactListViewProtocol: class {

    var presenter: ContactListPresenterProtocol?  { get set }

    /* Presenter -> ViewController */
    func showResponse(lists : [Contact])
}
```

Figure 21. Class Protocol VIPER

4.2 Quality

Below are result of experiment in case study contact using MVC, MVP, MVVM and VIPER in testability, modifiability and performance.

4.2.1 Testability

A. Size of test cases

In this architecture is ContactListVC class. The result comparison shows in Table 7.

Table 7: Lines test case each architecture

Architecture Pattern	Lines of code
MVC	164
MVP	129
MVVM	**113**
VIPER	123

MVC > MVP > VIPER > **MVVM**

MVVM has fewer line codes for testing because MVVM applies data binding in ViewModel. After each function that requires data, it will retrieve data in the ViewModel component. Unlike MVP and VIPER, data thrown into View will be processed and re-aligned in view. So, the declaration process occurs in each component. MVC has the most code because all aspects of the data, control view, handle a case, etc. are in the controller. The impact is that the longer the product development process, the number of lines in the controller component. making it more difficult to test.

B. Execution time to run application

Execution time build using XCode with clear cache in XCode before testing. The result of comparison execution time shows in Table 8.

Table 8: Differences time each architecture

Architecture Pattern	Time (Second)
MVC	24.1
MVP	20.9
MVVM	**19.6**
VIPER	21.4

MVVM has a testing time that is faster than other architectures. Because declarations in data usage are more divided into different components. On the other hand, MVC has a long-time due to a large number of business applications run within the Controller.

4.2.2 Modifiability

C. Cohesion Level

Modifiability criteria using Cohesion level procedural. Result cohesion shows at Table 9:

Table 9: Differences cohesion point each architecture

Architecture	Cohesion (Procedural)
MVC	0,4
MVP	0,3809
MVVM	**0,55**
VIPER	0,528

The result of cohesion each architecture shows in Table 10. MVP has the lowest value between 4 architectures. MVVM is the best of cohesion level in procedural.

D. Coupling Level

Modifiability criteria using coupling level data, and coupling level message.

The scenario for update new feature:

1. User click filter icon
2. Click event to Send to C/P/VM
3. Show loading state on view
4. Sending parameters to model / entity component, in this case – male/female
5. After a request, a response is delivered to C/P/VM.
6. When a response has feedback if the list contact filter successfully refreshes the view. If the list contact filter failed, show an error message. C/P/VM passes this command to View.
7. Dismiss loading state

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

The result coupling at Table 10:

Table 10: Coupling point each architecture

Architecture	Coupling (Message)	Coupling (Data)
MVC	N/A	0,0416
MVP	0,0208	0,0208
MVVM	0,0208	0,0312
VIPER	**0,0173**	**0,0173**

MVC does not have a coupling level message because the all of the business function application is handled by the Controller. Viper is the best architecture in coupling level messages and data, because the change of function involves three components.

C. Measuring Amount of Functionality

Amount of functionality based on class that extend UIViewController class. Weight FSM in this testing using Low because a total of functional number point no more than 50 attributes is low

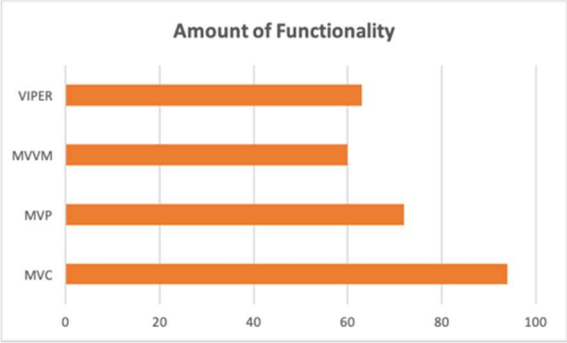

Figure 23. Class Protocol VIPER

Based on the result the highest count of functionality is MVC. Besides MVVM has the less of the amount of functionality

4.2.3 Performance

Performance divided in two criteria consumed memory and CPU. The following are the results of testability testing.

A. Consumed Memory

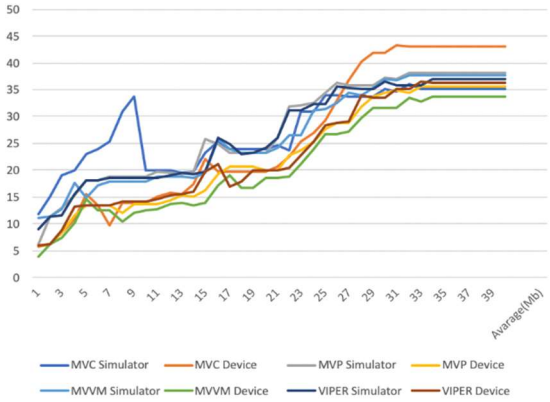

Figure 24. Consumed Memory

Table 11: Testing by simulator

Architecture	Average Memory (Mb)
MVC	27,7925
MVP	27,41225
MVVM	**26,4465**
VIPER	26,8085

Table 12: Testing by device

Architecture	Average Memory (Mb)
MVC	26,12875
MVP	22,9645
MVVM	**21,19275**
VIPER	23,269

Best performance in consumed memory in simulator and device is MVVM. Because when move from screen to other screen view the data model will be deleted and the ViewModel class will be deactivated. MVC, MVP, and VIPER have data declarations in the UIViewController class. When move from screen to other screen the data still remains in the UIViewController class.

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

B. Consumed CPU

Figure 25 shows the consumed CPU each architecture tested using simulator and device.

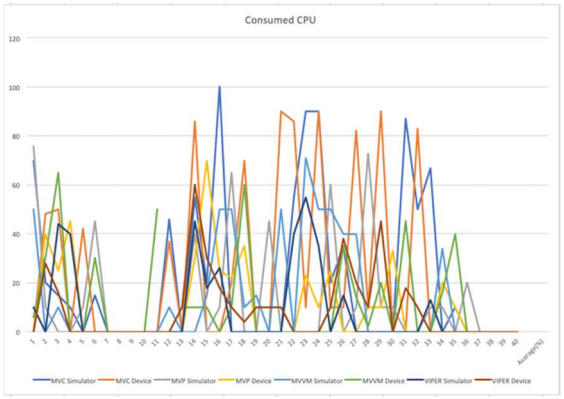

Figure 25. Consumed CPU

Table 13: Testing by simulator

Architecture	Average CPU (%)
MVC	21,375
MVP	12,6
MVVM	13,625
VIPER	**8,525**

Table 14: Testing by simulator

Architecture	Average CPU (%)
MVC	22,85
MVP	10,825
MVVM	11,61538462
VIPER	**8,925**

Best performance in consumed CPU in simulator and is VIPER. Because VIPER divided in 5 components and the functions in component not called simultaneously.

5. DISCUSSION

This section discusses the result on comparison of quality between MVC, MVP, MVVM and VIPER in Table 15 below.

Table 15: Comparison of Quality Architecture

Architecture Metrics	MVC	MVP	MVVM	VIPER
Testability				
Size of test cases	164	129	**113**	123
Execution time	24.1	20.9	**19.6**	21.4
Modifiability				
Cohesion	0,40	0,38	**0,55**	0,53
Coupling Message	N/A	0,0208	0,0208	**0,0173**
Coupling Data	0,041	0,0208	0,0312	**0,0173**
Amount of Functionality	94	72	**60**	63
Performance				
Performance CPU (Simulator)	21,3	12,6	13,6	**8,5**
Performance Memory (Simulator)	27,8	27,4	**26,4**	26,8
Performance CPU (Device)	22,8	10,8	11,6	**8,9**
Performance Memory (Device)	26,1	22,9	**21,1**	23,2

The result in **testability** show a good architecture based on fewer test cases and consumed less time to run a test.

Size of test case:

MVC > MVP > VIPER > **MVVM**

Consumed time:

MVC > VIPER > MVP > **MVVM**

Based on the criteria, the result of MVVM is the best of architecture in testability, and MVC is worse between four architectures were tested. Size of a test case in MVVM reduces because architecture implemented data binding in ViewModel which is each View using binding data from ViewModel than there is no need to declare data in a view.

The next result is on ~~good~~ which architecture has the best modifiability quality compared based on higher cohesion, lower coupling and less ~~of~~ amount functionality. In this case, the cohesion level used in this research is procedural, and coupling level used is Data and Message.

Cohesion Level (Procedural):

MVP < MVC < VIPER <**MVVM**

Coupling Level (Data):

VIPER < MVVM < MVP < MVC

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

Coupling Level (Message):

VIPER < MVVM = MVP

Amount of Functionality:

MVVM < VIPER < MVP < MVC

Results show that best architecture in modifiability cohesion level and amount of functionality is MVVM. Besides, best coupling level is VIPER. MVVM become best in cohesion because of MVVM reduce of task within a component connected by control connectors between View and ViewModel. Besides, VIPER become best on coupling because component in VIPER is the most than any other architecture.

The result of good architecture in **performance** based on less memory and CPU consumed.

Memory (Simulator):

MVC > MVP > VIPER > **MVVM**

Memory (Device):

MVC > VIPER > MVP > **MVVM**

CPU (Simulator):

MVC > MVVM > MVP> **VIPER**

CPU (Device):

MVC > MVP> MVVM > **VIPER**

The result shows that MVC architecture consumes most memory and CPU. Therefore, in device and simulator shows that VIPER is better that three-other architecture in CPU. Hence MVVM shows better than three other architectures for performance quality which is compared based on consumption of memory in device and simulator. Based on the quality results, MVVM and VIPER is identified as two of the best architecture, which has big potential in solving the problem of sustainable product in architecture pattern side. However, there are still opportunities in improving the architectures in the future.

6. CONCLUSION

This paper provides an interesting research on comparing mobile architecture pattern on 3 software qualities. Based on the findings, it seems MVVM and VIPER quite outstanding in term of testability, modifiability and performance. However, there a few things that needs to be considered to improve the quality of paper and hopefully close to be a candidate as an accepted paper.

The proposed future research might be the best practice to execute MVVM or VIPER architecture in iOS development. To gain better software qualities with migrating from the MVC to MVVM or VIPER architecture, which has the potential to become the basis for the development of mobile architectural patterns in the future.

The case study used in this paper is for proof of concept, more investigations should be carried using case study with higher complexity than the one used in this paper. Furthermore, third-party library affects the qualities of the software architecture. In this case, the similarities and differences, cons and pros used the third-party library. In addition, a measurement of the modifiability impact of refactoring from MVC or MVP to MVVM or VIPER in a complex case study is a good topic to be discussed.

7. ACKNOWLEDGEMENT

This research work has been carried out by Software Engineering Research Group (SERG), University Teknologi Malaysia (UTM) and is funded from Research University Grant (RUG) Universiti Teknologi Malaysia, under vote no. Q.J130000.2528.16H51. We would also like to thank Embedded & Real-Time Software Engineering Laboratory (EReTSEL) members for their feedback and continuous support.

REFRENCES:

[1] DeviceAtlas. (2018). The Mobile Web Intelligence Report, August 2018 Available from https://deviceatlas.com/blog/android-v-ios-market-share

[2] Kelly Rice, How Long Does It Take to Build a Mobile App. 2013, January 08. Available from https://www.progress.com/blogs/how-long-does-it-take-to-build-amobile- app.

[3] Habchi, S., et al. (2017). Code Smells in iOS Apps: How Do They Compare to Android? 2017 IEEE/ACM 4th International Conference on Mobile Software Engineering and Systems (MOBILESoft).

[4] Zabłocki Krzysztof (2017). Good iOS Application Architecture: MVVM vs. MVC vs. VIPER, May 8 Available from https://academy.realm.io/posts/krzysztofzabloc ki-mDevCamp-ios-architecture-mvvm-mvc-viper.

[5] Lou, T. (2016). A comparison of Android Native App Architecture MVC, MVP and MVVM, Aalto University. Master: 57.

[6] Soral, R. (2018). Ending the debate: MVC vs MVP vs MVVM for iOS application development, 10 January. Available from https://www.simform.com/mvcmvp-mvvm-ios-app-development.

[7] P. Bengtsson, N. Lassing, J. Bosch and H. v. Vliet (2000), "Analyzing software architectures for modifiability,". University of Karlskrona.

[8] Apple Documentation (2018). Cocoa Core Competencies: Model-View-Controller, November 2. Available from

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

https://developer.apple.com/library/archive/documentation/General/Conceptual/DevPedia-CocoaCore/MVC.html

[9] Mike Potel (1996). "MVP: Model-view-presenter the taligent programming model for C++ and java," Taligent Inc., p. 20, 1996.

[10] Gosman John (2005)." Introduction to Model/View/ViewModel pattern for building WPF apps" November 10. Available from https://blogs.msdn.microsoft.com/johngossman/2005/10/08/introduction-tomodelviewview model-pattern-forbuilding- wpf-apps.

[11] Duy, T. B. (2017). Reactive Programming and Clean Architecture in Android Development, Helsinki Metropolia University of Applied Sciences. Bachelor of Engineering Information Technology: 49.

[12] Syromiatnikov, A. and D. Weyns (2014). A journey through the land of model-viewdesign patterns. Proceedings - Working IEEE/IFIP 9Conference on Software Architecture 2014, WICSA 2014.

[13] X. Zhao, F. Khomh and Y. Zou, (2011). "Improving the Modifiability of the Architecture of Business Applications," in 2011 11th International Conference on Quality Software, Madrid.

[14] L. Bass, (2007). "Modifiability tactics," Carnegie- Mellon Univ Pittsburgh Pa Software Engineering Inst.

[15] Chen, M.-C. and Huang, S.-H. (2003) 'Credit scoring and rejected instances reassigning through evolutionary computation techniques', *Expert Systems with Applications*, 24(4), pp. 433–441.

[16] John M., Mike S. *iOS Design Patterns MVC and MVVM*, 2014, November 21. Available from https://www.captechconsulting.com/ blogs/ios-designpatterns- mvc-and-mvvm

[17] Torstensson, F. F. J. (2016). "Applying Mvp Principles When Developing Mobile Health Applications."

[18] TORSTENSSON, F. F. J. (2016). "Applying MVP Principles when Developing Mobile Health Applications."

[19] Laure, D., et al. (2016). Cross-platform development for Sailfish OS and Android: Architectural patterns and "dictionary trainer" application case study. 2016 19th Conference of Open Innovations Association (FRUCT).

[20] Harrison, N. B., et al. (2016). Software Architecture Pattern Morphology in Open-Source Systems. 2016 13th Working IEEE/IFIP Conference on Software Architecture (WICSA).

[21] Ojeda-Guerra, C. N. (2015). A simple software development methodology based on mvp for Android applications in a classroom context. Proceedings - 15th IEEE International Conference on Computer and Information Technology, CIT 2015, 14th IEEE International Conference on Ubiquitous Computing and Communications, IUCC 2015, 13th IEEE International Conference on Dependable, Autonomic and Secure Computing, DASC 2015 and 13th IEEE International Conference on Pervasive Intelligence and Computing, PICom.2015.

[22] Giedrimas, V. and S. Omanovič (2015). The impact of mobile architectures on component-based software engineering, Institute of Electrical and Electronics Engineers Inc.

[23] Putrama, I. M., et al. (2017). Architectural evaluation of data center system using architecture tradeoff analysis method (ATAM): A case study, Institute of Electrical and Electronics Engineers Inc.

[24] Saay, S. and A. Norta (2016). A reference architecture for a national e-learning infrastructure, Association for Computing Machinery, Inc.

[25] Mutual Mobile (2014). Meet VIPER: Mutual Mobile's application of Clean Architecture for iOS apps, September 24. Available from

[26] Felix Javier Acero Salazar, M. B. (2015). "Tailoring software architecture concepts and process for mobile application development." Proceedings of the 3rd International Workshop on Software Development Lifecycle for Mobile.

[27] Orlov Bohdan. (2015). 5 iOS Architectures in 5 minutes, 29 November. Available from http://slides.com/borlov/arch/fullscreen#/

[28] Larry D. Roper, K. A. R., J. Patrick Biddix (February 2018). Research Methods and Applications for Student Affairs, Jossey-Bass.

[29] Shah Hardik (2017). How to improve your mobile app's performance, August 18 Available from https://www.simform.com/mobile-app-performance/

[30] Pham Khoa (2018). A taste of MVVM and Reactive Paradigm, August 16 Available From https://flawlessapp.io/blog/a-taste-of-mvvm-and-reactive-paradigm

[31] Fineberg Alan (2015). Ziggurat iOS App Architecture, December 19 Available from https://medium.com/square-corner blog/ziggurat-ios-app-architecture b54b3f7132f0

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

[32] Novoseltseva Ekaterina (2017). Viper architecture advantages for ios apps, June 13 Available from https://apiumhub.com/tech-blog-barcelona/viper-architecture/

[33] B. Baudry and Y. Le Traon, "Measuring design testability of a UML class diagram," Information and software technology, vol. 47, pp. 859--879, 2005.

[34] F. Buschmann , R. Meunier, “Patternoriented software architecture: a system of patterns”, John Wiley & Sons, Inc., 1996

[35] Gallagher, K., et al. (2008). "Software Architecture Visualization: An Evaluation Framework and Its Application." IEEE Transactions on Software Engineering 34(2): 260-270.

[36] Lague, B., et al. (1998). An analysis framework for understanding layered software architectures. Proceedings. 6th International Workshop on Program Comprehension. IWPC'98 (Cat. No.98TB100242).

[37] Latum, F. V., et al. (1998). "Adopting GQM based measurement in an industrial environment." IEEE Software 15(1): 78-86.

[38] Keith. A and Arron J.H (2019) “Random User Generator” from https://randomuser.me/