# Evaluation of Software Design Pattern on Mobile Application Based Service Development Related to the Value of Maintainability and Modularity

Billy Susanto Panca
Faculty of Information Technology
Maranatha Christian University
Bandung, Indonesia
Email: billy.sp@it.maranatha.edu

Sukrisno Mardiyanto
School of Elect. Eng. & Inf.
Bandung Institute of Technology,
Bandung, Indonesia
Email: sukrisno@stei.itb.ac.id

Bayu Hendradjaya
School of Elect. Eng. & Inf.
Bandung Institute of Technology,
Bandung, Indonesia
Email: bayu@stei.itb.ac.id

***Abstract*-This paper presents an implementation of software design pattern on development of mobile application based services. The main purpose of this research is to find and to evaluate combination of software design pattern related to the value of maintainability and modularity. Combination of design patterns that used in this research is singleton, memento, state, iterator, factory, builder, and flyweight. Three mobile apps used for case studies are m-Learning, m-Health, and m-Survey. Each of case study has different domain such as m-Learning are many uses of multimedia content, m-Health which is more inclined to use of algorithm, and m-Survey more use of plain data. These three case studies use the "thick client – thin server" concept. Selection of these case studies is intended to represent the characteristics of mobile application development domain. These three case studies are running on Android platform. Implementations and measurements are done gradually for each selected design pattern.**

***Keywords : Mobile Application Development; Android; Design Pattern; Maintainability; Modularity; Quality Metrics; SOA***

## I. INTRODUCTION

Development of software included mobile application domain usually just focus on its functionality and doesn't care about how to maintain and evolve it. The development process that only focused on functionality in general be built using the concept of anti-pattern. The next issue is about the value of maintainability and modularity of the applications, the concept of anti-pattern can complicates the process of maintenance and software evolution [1]. Software development must consider about maintainability and modularity [2] [3].

Now day, mobile application development mostly use Service Oriented Architecture (SOA) for increasing maintainability value in content management by putting content on server side and give it to client when requested. Benefits of using SOA is makes the software become loosely coupled between logic and data, because when the content is changed, the apps doesn't need to update the code [4] [5] [6] [7].

Besides the benefits of using SOA, there are challenges by using it. The used of SOA with "thick client – thin server" concept give challenge in maintain process [8]. Thin concept based on consideration that client hardware can independently process data and algorithm by it self, so that client just only need data provided by server. This method can save servers resources and make it can handle more client rather than making process for several clients which can actually be done by the client itself.

Besides the challenges of SOA, there are another challenges while developing mobile application by using object oriented programming (OOP) such Android. The OOP concepts can degrade the quality of the program code if there is a used of god class and object orgy concept as anti-pattern. [1]. In addition, the dependence between objects is also a factor to be considered because the more amount of dependencies between objects means the more difficult to maintained in the future [9].

Implementation of software design patterns in OOP-based application development has been considered can improve maintainability and modularity [2]. Nonetheless, Android mobile application development based OOP is different from desktop application development, such as application life cycle, procedures for the use and access of resources between object, the concept of multi-threading, and layout object creation process [10]. It makes the used of software design patterns in the development of Android mobile applications need to be investigated whether it can improve maintainability and modularity metric, or make the code become more complex. Implementation of design patterns can effect good or bad depending on the combination of design patterns and correct use of the issue to be resolved [2].

## II. RELATED WORK

Many related work used a mobile app-based services and mobile applications using software design patterns. Here we take the case studies within domain mobile learning, mobile survey, and expert system by using the service as a content provider [11] [12] [13]. A similar research using case studies in these domains focus on the implementation of the method used for this application to run as needed to function and is not concerned with aspects of maintainability and modularity of application although it is important to keep in mind as the mobile application development process.

In this study, we focused on evaluating the combination of software design patterns on mobile applications development based service with anti-pattern basis, and measure the maintainability and modularity value before and after the implementation of software design patterns.

## III. Mobile Application Development Characteristics

There is several characteristics at mobile application development such execution unit life cycle (well known as Activity class in Android), data passing between Activity using Intent object or global variable, build layout object statistically using XML schema or dynamically using OOP, multi threading using asynchronous task, and resource management [10] [14]. Since mobile application can be build by object oriented programming such Android using Java language, developing mobile apps is different with desktop apps due to differences in the procedures for setting and using resources by operation system.

There are four execution layer in Android development. Grouping these layer is intended to reduce coupling [15], but the using of anti-pattern concept can make it hard to maintained and modified. Class dependency between packages need to be regulated because if not it will make an application be not modular to redeveloped.

Mobile application service based development also have problems to keep in mind. Such as designing how to initialize a connection to the service, dependence of service when the connection is lost, and the procedures for handling the types of data exchanged between client-server [16]. These kind of problems need to be maintained and modified in the future, so that application development process should be set to ease the ability to adapt the changes.

## IV. Software Design Patterns for Enhance Maintainability and Modularity

To say that an application is easy to maintained and modified we need a measurement figures. The use of formulas maintainability [17] and modularity [18] can provide value that can be used as a reference measurement. Software design pattern said can increase the value of maintainability and eases the process of evolution [2] [9].

Software design pattern can be used at mobile application development since it using object oriented programming principle and design pattern extend object oriented programming itself [2]. To evaluate that implementing software design pattern would increase or decrease value of maintainability and modularity index, we prepare three case studies with anti pattern such as god class [18] [19] and also using four execution layers such as MVC from the baseline case studies [20].

Before implementing software design pattern, the case studies project using anti pattern will be measured its maintainability [17] and modularity [18] metric first, and we try to measure how much effect of implementing design pattern in mobile development based service development. Several software design pattern selected by categories of patterns such creational, behavioral, and structural to measure the changes of maintainability and modularity metric.

Maintainability be calculated by measuring metrics such halstead volume, non-comment line of code, and cyclomatic complexity [17]. There is one more metric included in maintainability formula that is comment line of code, but this metrics will not included in calculation since of there are not a lot of appearance in case studies that we used.

Different from the method of calculating maintainability which just use class level, formula of modularity metric calculated in step-by-step level from class level, package level, and system level to measure modularity index. Class level modularity can be calculated by using basic metrics such as non-comment line of code, number of operation, and lack of cohesion. Average of class level in a package will be package level modularity. System level can be calculated by getting package cohesion and package coupling. Based on these levels we can calculate the modularity index [18] [21].

## V. The Merits of Using Software Design Patterns

Since there are classifications of software design pattern related to application such as creational pattern, behavioral pattern and structural pattern we try to choose several pattern to be implemented based on case studies. The following case studies such m-Learning, m-Health, and m-Survey will be used to measure the impact of usage of the design patterns.

Every case studies have their own functionalities and characteristics. M-Health application used to calculate expert system algorithm, m-Learning application use a lot of multimedia content, and m-Survey application that many exchanging data with web service application. These case studies are expected to represent the entire mobile application.

We propose seven design patterns such as singleton, memento, state, iterator, factory, builder, and flyweight pattern. These patterns will be implemented step-by-step and measure on each case studies. Baseline version will be the used no pattern, and continued adding design patterns with in the following order:

- Version–1 : + Singleton Pattern
- Version–2 : + Memento Pattern
- Version–3 : + State Pattern
- Version–4 : + Iterator Pattern
- Version–5 : + Factory Pattern
- Version–6 : + Builder Pattern
- Version–7 : + Flyweight Pattern

The selection of software design patterns based on the characteristics of mobile application development that has been described previously. Singleton pattern is used to ease the process of passing data between application life cycle. Memento and state pattern are used to handle mobile application life cycle such as activity object and layout fragmentation on Android application. Iterator, factory, builder, and flyweight pattern are used to handle object creation. Iterator used for the creation of objects repeatedly while hiding the complexity of the object. Factory pattern used for layout object creation on all case studies. Builder pattern used to create a connection object between client-server, and also used on each applications based on the functionality of each application such as creating 3D object on m-Learning application using augmented reality, create expert system result on m-Health application, and image captured manipulation using camera on m-Survey application.

Flyweight pattern used for creating many object with the same behavior such as text view on m-Learning application.

Figure 1 shows the connections between the selection of 7 design patterns applied to the characteristics of mobile applications of each package in the application.

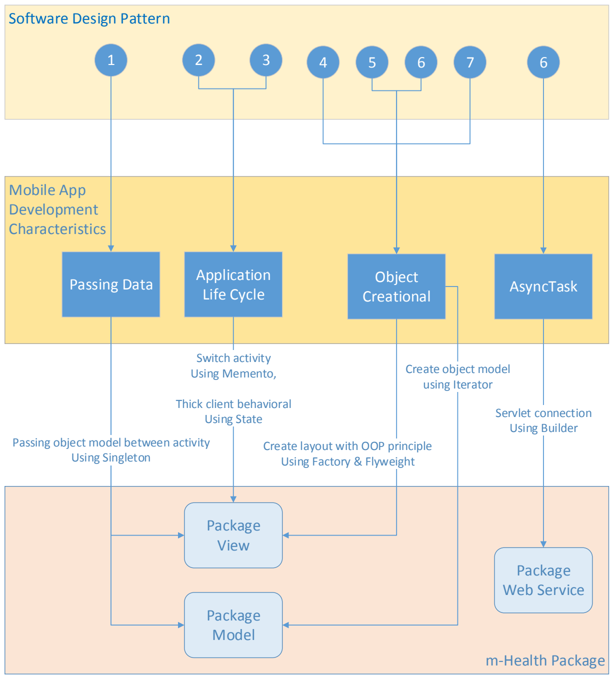
Software Design Pattern
1
2
3
4
5
6
7
6
Mobile App Development Characteristics
Passing Data
Application Life Cycle
Object Creational
AsyncTask
Switch activity Using Memento,
Thick client behavioral Using State
Create object model using Iterator
Servlet connection Using Builder
Passing object model between activity Using Singleton
Create layout with OOP principle Using Factory & Flyweight
Package View
Package Model
Package Web Service
m-Health Package

Fig. 1. Software design pattern implementation flow on m-Health application

For each version of the use of software design patterns, the following results are applied in the Table I and II. The calculation of maintainability and modularity carried out step by step for each version of case studies.

TABLE I. MAINTAINABILITY MEASUREMENT

| **Version** | **Case Studies** | | |
|---|---|---|---|
| | ***m-Health*** | ***m-Learning*** | ***m-Survey*** |
| Baseline | 61.7361094 | 58.07116858 | 47.58032129 |
| Version 1 | 62.5434490 | 58.64638713 | 48.04872818 |
| Version 2 | 65.4954434 | 61.17704853 | 51.00381175 |
| Version 3 | 68.1785687 | 66.84899707 | 54.16771413 |
| Version 4 | 70.8058285 | 68.13499165 | 56.36948175 |
| Version 5 | 75.6369094 | 70.65762474 | 60.58250202 |
| Version 6 | 80.0302681 | 73.35470044 | 64.64117111 |
| Version 7 | 80.1244708 | 73.16787573 | 64.56086496 |

Meaning of maintainability index is to show the low and high ability of the application to be maintained.

High Maintainability : MI >= 50 and

Low Maintainability : MI < 50 [22].

Based on the results of the measurement, maintainability index statistically continues rise along with addiction of software design pattern that have been selected to use. Figure 2 shows the increase in the value of maintainability in the initial version to the final version.

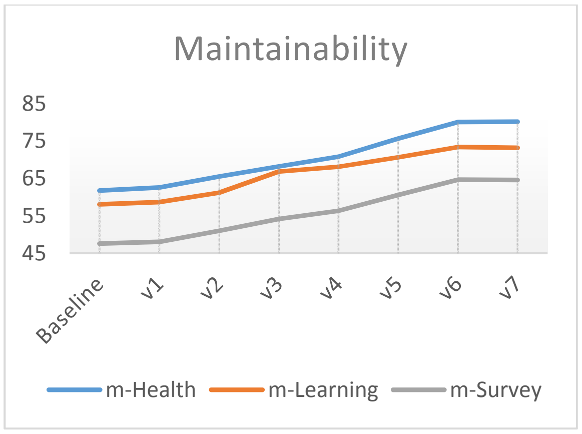

Fig. 2. Maintainability Measurement Chart

The use of chosen software design pattern makes application module size larger, because of increasing the number of classes and packages. However, design pattern makes base classes redistribute LoC, operand and operator, and cyclomatic complexity on each class to design pattern classes. It makes average of line of code, halstead volume, and cyclomatic complexity reduced because taking the average metrics of all classes in the module of application development. However, using flyweight pattern doesn't increases index significantly because it doesn't added a new class like others.

Besides calculate the value of maintainability, we also calculate the modularity values. Table II shows the results of applying software design patterns on every version of the case studies.

TABLE II. MODULARITY MEASUREMENT

| **Version** | **Case Studies** | | |
|---|---|---|---|
| | ***m-Health*** | ***m-Learning*** | ***m-Survey*** |
| Baseline | 0.072955031 | 0.090812354 | 0.098554527 |
| Version 1 | 0.048895465 | 0.081354078 | 0.092844038 |
| Version 2 | 0.042701388 | 0.079432841 | 0.082364407 |
| Version 3 | 0.070138680 | 0.083975686 | 0.083021220 |
| Version 4 | 0.054954864 | 0.076756791 | 0.069100670 |
| Version 5 | 0.105313027 | 0.073968293 | 0.063777100 |
| Version 6 | 0.103883388 | 0.079506965 | 0.064928044 |
| Version 7 | 0.103883464 | 0.078315305 | 0.064928041 |

Different with result on maintainability which give significant index increases, implementation of software design patterns on the side of modularity makes it go up and down for each pattern. From the figure. 3, m-Learning and m-Survey case studies, the using of software design pattern make its modularity drops, except while using state pattern. M-Health app have a similar result, but there is another increases in modularity value while using factory pattern. It because in the

M-Health app that just using small number of classes in the view package that effectively uses the factory pattern to create layout object using OOP concept while M-Learning and M-Survey combine OOP concept and XML schema to build layout component on view package.

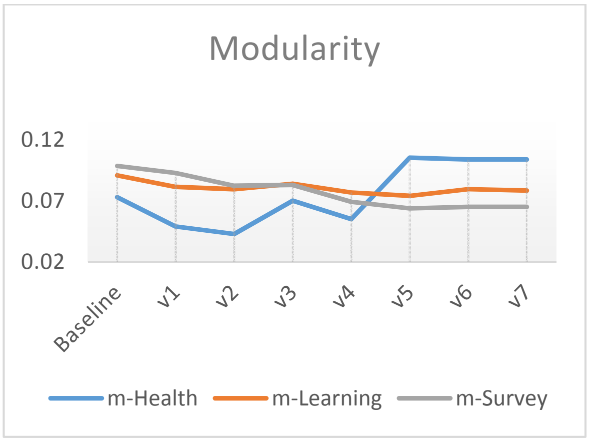

Fig. 3. Modularity Measurement Chart

Because of calculation modularity metric start by calculating values of class level, package level and system level, the structure of dependencies between classes in a package and classes in another package gives significant impact for the value of modularity. The use of design pattern change base class structure and also add pattern classes and packages. Besides of reducing average of LoC, LCOM, and number of function it also make many dependencies to a new classes of software design patterns. It causes an increase in the value of package coupling that reduces the value of the system architecture.

TABLE III. SYSTEM ARCHITECTURE MEASUREMENT

| Version | Case Studies | | |
|---|---|---|---|
| | *m-Health* | *m-Learning* | *m-Survey* |
| Baseline | 0.171860016 | 0.338794524 | 0.337015463 |
| Version 1 | 0.120385853 | 0.294479968 | 0.336341992 |
| Version 2 | 0.102514861 | 0.226126416 | 0.218775458 |
| Version 3 | 0.145517881 | 0.197479544 | 0.188221463 |
| Version 4 | 0.116071712 | 0.180590293 | 0.157542407 |
| Version 5 | 0.210086153 | 0.176981916 | 0.136041858 |
| Version 6 | 0.188375291 | 0.169237694 | 0.125311516 |
| Version 7 | 0.188375291 | 0.166623657 | 0.125311516 |

Table III show the value of system architecture for every version. The system architecture values decreased with the implementation of software design pattern. For all the selected design patterns, they are increase the value of package coupling rather than increase package cohesion. The modular software must have a "low coupling, high cohesion" value.

The modularity index decreases with a decrement of system architecture value on the application after the implementation process of design patterns. Figure 4 shows the similarity of decrement on system architecture value and modularity index.

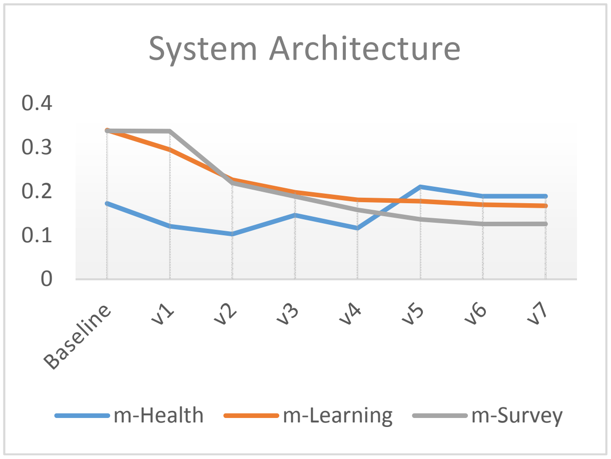

Fig. 4. System Architecture Measurement Chart

## VI. CONCLUSION AND FUTURE WORK

The proposed of software design patterns can affect the value of the results obtained. Based on statistically examination results, the maintainability metrics value is increased while using software design patterns compared while using anti-pattern. However, the results are not aligned. The value of maintainability increase steadily, compared with modularity which is not.

Combination of software design patterns that good to implemented on mobile application service based development related to increases of maintainability. However, the using of flyweight didn't give big impact to the value of maintainability.

The evaluation result showed that the using of proposed software design patterns in mobile applications can reduce the value of modularity, except the state pattern. Software design pattern such as singleton, memento, and iterator reduce the value of modularity in all the case studies. This is due to the increase of package coupling value and reduction the package cohesion value while using the patterns.

The factory and builder pattern can increase and reduce the value of modularity depends on case studies. The m-Health application have a little number of view classes that using factory and builder. M-Health also just use OOP concept to create layout object, so then factory and builder pattern can effectively used. Rather than m-Learning and m-Survey that have a many classes in view package that combine OOP concept and XML schema to create object layout. Factory and builder pattern have a high package coupling and affect the average of whole application module depends how we use those patterns. The factory pattern is fit to use for increasing modularity value while the apps using OOP concepts to create object layout besides using XML schema.

In fact, we need to implement a method that can facilitate the process of maintain and evolution. However, the use of design pattern can improve maintainability also lowered modularity value. We need to consider the using of design patterns for the mobile application development problems. The proposed design patterns that used to solve faced problem seems effective for increase maintainability. Although it can increase the value of package coupling without increase in package cohesion and reduction in the value of modularity.

We need to combine more software design patterns that can make the development process has a good value of maintainability and modularity value. Also the addition of case studies with larger scale so that the more result can be obtained and analyzed.

## REFERENCES

[1] S. Kaur and S. Singh, "Influence of Anti-Patterns on Software Maintenance: A Review," *ICAET,* vol. 8887, p. 975, 2-15.

[2] E. Gamma, Helm, Richard, R. Joshson and J. Vlisides, Design Pattern - Object of Reusable Object Oriented Software, 1995.

[3] M. Zneika, H. Loulou, F. Houacine and S. Bouzefrane, "Toward a Modular and Lightweight Model for Android Development Platform," in *IEEE*, Paris, France, 2013.

[4] L. Dikmans and R. van Luttikhizen, SOA Made Simple (8), 2012.

[5] T. Erl, SOA Principles of Service Design, Prentice Hall, 2007.

[6] M. Kaufmann and D. K. Barry, Web Services and Service Oriented Architecture : The Savvy Manager's Guide, Elsevier Science, 2003.

[7] G. zeiss, "Interoperability using Tightly and Loosely Coupled (SOA) Architecture," 2006.

[8] M. R. Izadi, "Fat/Thin client for mobile applicationsand the proposed way forward," KTH Information andCommunication Technology, Stockholm, Sweden, 2012.

[9] C. R. Martin, "Design Principles and Design Patterns," 2000.

[10] M. L. Murphy, Beginning Android 2, Apress, 2010.

[11] M. G. Anshumati and M. K. Shrikant, "Conceiving M-Learning Application: A Step towards a Interactive Omnipresence Environment in Android based mobile," *International Journal of Engineering Science Invention,* vol. 2, no. 4, pp. 07-13, 2013.

[12] M. M. Ali, A. F. Elsharkawi, M. G. El Said and M. Zaki, "Design Pattern for Multimedia Mobile Application," *Journal of Computer Science And Software Application,* vol. 1, no. 2, 2014.

[13] S. M and D. T. A, "Designing An M-Learning Application for a Ubiquitous Learning Environment in The Android Based Mobile Devices Using Web Service," *Indian Journal of Computer Science and Engineering,* vol. 2, no. 1.

[14] D. Griffiths and D. Griffith, Head First Android Development, O'Reilly.

[15] W. Yong Kim and S. Gyu Park, "The 4-Tier Design Pattern for the Development of an Android Application," *Future Generation Information Technology,* vol. 7105, pp. 196-203, 2011.

[16] M. Husain Bonara, M. Mishara and S. Chaudary, "RESTful Web Service Integration using Android Platform," in *IEEE*, 2013.

[17] D. W. Kurt, The Software Maintainability Index Revisited, 2001.

[18] A. W. Rahadjo and D. J. Surjawan, "Revised Modularity Index to Measure Modularity of OSS Projects with Case Study of Freemind," *International Journal of Computer Applications,* vol. 59, p. 0975 – 8887, 2012.

[19] W. H. Brown, C. R. Malveau and T. J. Mowbray, Anti Pattern - Refactoring Software, Architectures, and Project in Crisis, Canada: Robert Ipsen, 1988.

[20] Hyun Jun La and Soo Dong Kim, "Balanced MVC Architecture for Developing Service-based Mobile Application," 2010.

[21] A. W. Rahardjo and R. W. , "Modularity Index Metrics for Java-Based Open," 2011.

[22] F. R. Oppedijk, "Comparison of the SIG Maintainability Model," 2008.