THESIS – KI142502

THE IMPACT OF ENTERPRISE SOFTWARE DESIGN PATTERNS ON MAINTAINABILITY METRICS: A CASE STUDY

I MADE BHASKARA GAUTAMA
NRP. 5114201017

SUPERVISORS
Dr. Ir. Siti Rochimah, MT.
Rizky Januar Akbar S.Kom., M.Eng.

MASTER PROGRAM
AREAS OF EXPERTISE: SOFTWARE ENGINEERING
DEPARTMENT OF INFORMATICS
FACULTY OF INFORMATION TECHNOLOGY
INSTITUT TEKNOLOGI SEPULUH NOPEMBER
SURABAYA
2018

THESIS APPROVAL SHEET

A thesis submitted in partial fulfillment of the requirements for the degree of

Magister Komputer (M.Kom.)

Institut Teknologi Sepuluh Nopember Surabaya

by

I MADE BHASKARA GAUTAMA

NRP. 511420117

Title:

The Impact of Enterprise Software Design Patterns on Maintainability Metrics: A Case Study

Date of Seminar : January 4, 2018

Graduation Period : 2018 Even Semester

Approved/Certified by:

Dr. Ir. Siti Rochimah, M.T.	[illegible]
NIP. 196810021994032001	(Supervisor 1)
Rizky Januar Akbar, S.Kom, M.Eng.	[illegible]
NIP. 198701032014041001	(Supervisor 2)
Daniel Oranova Siahaan, S.Kom, M.Sc, PD.Eng.	[illegible]
NIP. 197411232006041001	(Examiner 1)
Dr. Eng. Chastine Fatichah, S.Kom, M.Kom.	[illegible]
NIP. 197512202001122002	(Examiner 2)
Dr. Eng. Darlis Heru Murti, S.Kom, M.Kom.	[illegible]
NIP. 197712172003121001	(Examiner 3)

KEMENTERIAN RISET, TEKNOLOGI, DAN PENDIDIKAN TINGGI
INSTITUT TEKNOLOGI SEPULUH NOPEMBER
FAKULTAS TEKNOLOGI INFORMASI DAN KOMUNIKASI

Dean of the Faculty of Information Technology,

[illegible]

Dr. H. Agus Zainal Arifin, S.Kom., M.Kom.

NIP. 197208091995121001

[This page is intentionally left blank]

THE IMPACT OF ENTERPRISE SOFTWARE DESIGN PATTERNS ON MAINTAINABILITY METRICS: A CASE STUDY

Name	: I Made Bhaskara Gautama
Student Identity Number	: 5114201017
Supervisor	: Dr. Ir. Siti Rochimah, MT.
Co-Supervisor	: Rizky Januar Akbar S.Kom., M.Eng.

ABSTRACT

Design pattern in software engineering is a set of solutions that is used to solve software development common problems. The purpose of using design patterns is to improve software quality. There are several design patterns that have been proposed. One of them is enterprise design patterns which are specified for enterprise application. However, there lack of literature that studies these patterns.

This study proposes a methodology to measure the impact of design patterns on software maintainability attribute. It uses Academic Information System (AIS) of Institut Teknologi Sepuluh Nopember (ITS) as a case study. It is an enterprise application because it involves persistent data. It was built without using any standards, thus becomes more complex along with how often the maintenance is conducted. It indicates that AIS has a low maintainability. Based on ISO/IEC 25010, maintainability is one of the quality attributes. It uses C&K metrics and three additional metrics to measure the maintainability. Measurements are conducted on software which is built without considering the use of design patterns and software which is built using design patterns. Both of the results are evaluated to obtain scientific evidence of the impact.

There are two pattern versions produced in this study. Both of the pattern versions are able to improve the maintainability. We evaluate the impact based on the layer. On presentation layer, pattern versions are able to improve the maintainability to a small extent. On domain layer, it is improved to a certain extent. On data-source layer, it is improved to a great extent. Pattern versions are also able to decrease the number of duplicated methods.

***Keywords*:** *Design patterns, maintainability, refactoring, enterprise, academic information system.*

[This page is intentionally left blank]

PREFACE

First of all, I would like to thank God for being able to finish the thesis report titled “The Impact of Enterprise Software Design Patterns on Maintainability Metrics: A Case Study“.

This thesis is a final work as a partial fulfillment for the Master degree at Institut Teknologi Sepuluh Nopember Surabaya. The completion of this report cannot be separated from the support of various people. They have contributed either directly or indirectly to this study which makes me very grateful. But most of all, I would like to thank both of my supervisors, Mrs. Dr. Ir. Siti Rochimah, MT. and Mr. Rizky Januar Akbar S.Kom., M.Eng for their helpful comments, suggestion, feedback, and so forth. I also would like to thank all the people who have supported me so far. Thank you for your encouragement and tireless enthusiasm.

Needless to say, this thesis report is far from perfect. Therefore, I really hope for helpful suggestions and advices to improve this report. At last, I hope this report can be useful for everyone.

Surabaya, January 8, 2018

Author

[This page is intentionally left blank]

TABLE OF CONTENTS

TABLE OF FIGURES

TABLE OF TABLES

CHAPTER I
INTRODUCTION

1.1 Background

Design patterns are used to improve software quality. The most famous and well-developed software design patterns are Gang of Four (GoF) design patterns: Gamma, Helms, Johnson, and Vlissides (Gamma et al. 1994). Research of software design patterns in various fields is still conducted until nowadays. There are several design patterns which are proposed, i.e., GoF 1994, Buschmann 1996, Serial 2011, Sinha 1996, Fowler 2002, and so on (Gonzalez-Sanchez et al. 2012). Design patterns consist of a set of solution. They are used to solve software development common problems. Thus, they shorten the software development, reduces costs, and improve the software quality (Ali & Elsih 2013; Christopoulou et al. 2012). Usually, design patterns cannot be used directly into the source code because it is a description or template. It is used to guide the software development to produce a more reusable code.

This study uses Academic Information System (AIS) as a case study. It is an AIS of Institut Teknologi Sepuluh Nopember (ITS). It is an enterprise system that is operated to ease long-term student academic administration. AIS is often maintained due to changes in business process, standard operating procedure, and features. Maintenance process is difficult because one change in certain code affects another code in several places. Doing this process repetitively may increase structure complexity of the software. Thus, the future maintenance process will be difficult and likely impossible to do. High coupling value causes this problem occurs. It indicates that the software has low modularity. Thus, it affects maintainability as well.

Refactoring is a technique to handle this problem. It changes the internal structure without affects the external function (Muraki & Saeki 2002). This study involves the application of design patterns to lead the software refactoring. We use enterprise software design patterns by Fowler (Fowler et al. 2002). The main reason of utilizing those patterns is because AIS involves persistent data. Applying design patterns aim to improve the software maintainability.

While AIS is being developed, it involves two teams. The original AIS was built by the first team, while the second team is performed the maintenance and improvement. It was done without using any standards and not all of AIS is well managed (Rochimah et al. 2014). Based on the result of research, AIS needs to be evolved (Rochimah et al. 2015). AIS structure becomes more complex along with how often the maintenance is conducted. It is one of the challenges in software evolution that is software erosion (Handani & Rochimah 2015). Thus, it needs a re-engineering to fix this problem. It involves design patterns to improve the software maintainability. There are so many design pattern literature which may be used for references. However, the impact of design patterns on software quality attributes are still controversial (Ampatzoglou et al. 2013; Ali & Elsih 2013). There are also lacks of literature that studied enterprise design patterns specifically. So, we propose a quantitative research to study the impact of enterprise design patterns on software maintainability. We measure the software maintainability by using software quality metrics (Muraki & Saeki 2002; Ampatzoglou et al. 2012). We use international standard ISO/IEC 25010 and ISO/IEC 25023 as well (ISO/IEC 25010 2011; ISO/IEC 25023 2015). We also investigate the code changes that occurred when feature is added. The purpose of this study is to produce scientific evidence in which design patterns may improve the software maintainability. The result of this study is expected to help developer to determine appropriate patterns when conducting re-engineering on AIS.

1.2 Research Questions

Our hypothesis is constructed in the following sentence: “Utilizing design patterns may improve the software maintainability”. This study uses three research questions to formulate the problem. The questions are stated as follows.

1. To what extent application of design patterns may affect software maintainability?

2. How to indicate appropriate patterns on a selected case based on some specific criteria?
3. How to validate the impact of utilizing design patterns on software maintainability?

1.3 Purpose and Significance

The purpose of this study is to investigate to what extent the design patterns affect software maintainability.

The benefit of this research is to provide design patterns recommendation that used to lead AIS re-engineering process. Thus, the software is easier to maintain for long term maintenance processes.

1.4 Limitations

This study limits the problems to prevent it be widened. Limitations that are used in this study are organized as follows.

1. Using AIS (research version, which is built using Java) of ITS as a case study.
2. Focusing on software maintainability measurement.
3. Using enterprise software design patterns by Martin Fowler (Fowler et al. 2002).
4. Using Java programming language.

[This page is intentionally left blank]

CHAPTER II
LITERATURE STUDY

Basic theories that used to solve the problems discusses in this chapter. Theories include design patterns for enterprise object-oriented application architecture, software quality, measurement method, evaluation method, and academic information system of ITS.

2.1 Enterprise Design Pattern

Based on Martin Fowler's book "Patterns of Enterprise Application Architecture" (Fowler et al. 2002), there are distinct kinds of software application. One of those is enterprise application. It is usually involves a lot of persistent data, concurrently accessed by many people, and a lot of user interface to handle the data. Also, it usually integrates with other enterprise application. Table 2.1 shows types and names of enterprise design patterns.

Table 2.1 Enterprise design patterns (Fowler et al. 2002).

Numb	Types	Patterns
1	Domain Logic Patterns	Transaction Script
		Domain Model
		Table Module
		Service Layer
2	Data Source Architectural Patterns	Table Data Gateway
		Row Data Gateway
		Active Record
		Data Mapper
3	Object-Relational Behavioral Patterns	Unity of Work
		Identity Map
		Lazy Load
4	Object-Relational Structural Patterns	Identity Field
		Foreign Key Mapping
		Association Table Mapping
		Dependent Mapping
		Embedded Value
		Serialized LOB
		Single Table Inheritance
		Class Table Inheritance
		Concrete Table Inheritance
		Inheritance Mapper
5	Object-Relational Metadata Mapping Patterns	Metadata Mapping
		Query Object
		Repository

6	Web Presentation Patterns	Model View Controller
		Page Controller
		Front Controller
		Template View
		Transform View
		Two Step View
		Application Controller
7	Distribution Patterns	Remote Facade
		Data Transfer Object
8	Offline Concurrency Patterns	Optimistic Offline Lock
		Pessimistic Offline Lock
		Coarse-Grained Lock
		Implicit Lock
9	Session State Patterns	Client Session State
		Server Session State
		Database Session State

This sub chapter only mentions design patterns which are used in the preliminary study which are Domain Model, Active Record, and Data Mapper.

1. Domain Model

   Domain Model is an object model of the domain that incorporates both behavior and data. Business logic of the software can be very complex. Rules and logic describe many different cases. Objects were designed to deal with this complexity. A Domain Model creates connections of interconnected objects, where each object has its own functions and behavior. Figure 2.1 shows the example of Domain Model class diagram (Fowler et al. 2002).

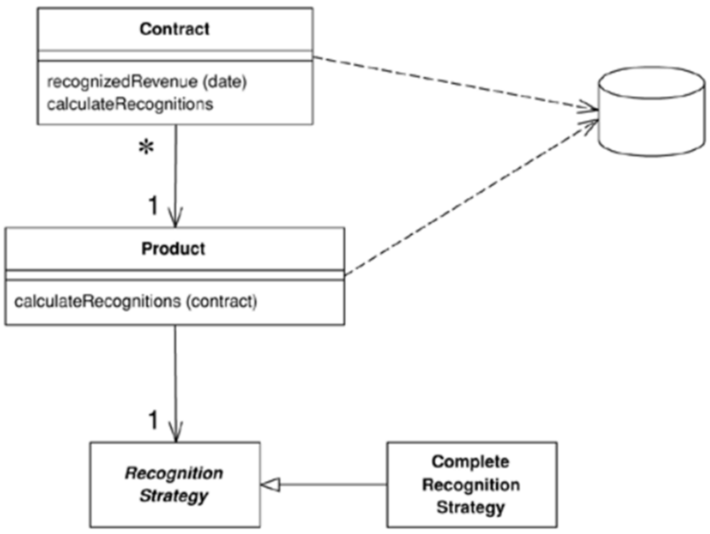


Figure 2.1 Domain model class diagram.

Using Domain Model in an application involves inserting a whole layer of objects. We work on business area which is modeled by these objects. Objects represent the data and capture the rules that are used in the business. Mostly, the data and processes are combined to make them work together.

Domain model is used if there are a complicated and ever changing business rules involving validation, calculation, and derivation. Aside from that, if there are a simple not null checks and a couple of sums to calculate, then a Transaction Script is a better solution.

2. Active Record

Active Record is an object that wraps a record data structure in an external resource, such as a row in a database table, and adds some domain logic to that object. An object carries both data and behavior. Much of this data is persistent, and needs to be stored to a database. Active Record put the data access logic into the domain object. Figure 2.2 shows the example of Active Record class diagram (Fowler et al. 2002).

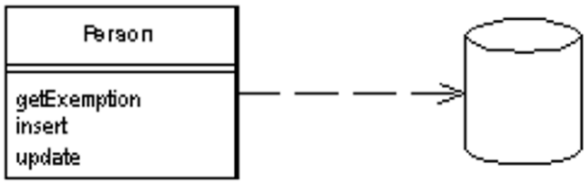


Figure 2.2 Active record class diagram.

Active Record can be used together with Domain Model. This is because the classes in Domain Model match very closely with the record structure of an underlying database. Each active record is responsible to saving nd loading to the database, and also any domain logic that acts upon the data. The data structure of the Active Record should exactly match that of the database that is one field in the class for each column in the table. The Active Record class typically has the following methods:

- Construct an instance of the Active Record from a SQL result set row.
- Construct a new instance for later insertion into the table.
- Static finder methods to wrap commonly used SQL queries and return Active Record objects.

- Methods to update the database and insert into the database with the data in the Active Record.
- Getting and setting methods for the fields.
- Methods that implement some pieces of business logic.

Active Record is a good choice when your domain logic is not too complex, such as create, read, update and deletes. Derivations and validations based on a single record work well in this structure. In an initial design for a Domain Model the main choice is between Active Record and Data Mapper. Active Record has the primary advantage of simplicity. It is easy to build Active Records and they are easy to understand. The primary problem with them is they work well only if the Active Record objects correspond directly to the database tables, an isomorphic schema. If the business logic is complex then it leads us to use the object mechanisms such as direct relationships, collections, and inheritance. These do not map easily onto Active Record. Adding them piecemeal soon gets very messy, so it will lead the use of Data Mapper.

3. Data Mapper

Data Mapper transfers data from a domain object to a database. Objects and relational databases have different mechanisms for structuring data. Many parts of objects, such as collections and inheritance are not present in relational databases. When building an object model with a lot of business logic, it is valuable to use these mechanisms to better organize the data and the behavior that goes with it. This leads to variant schemas, where the object schema and the relational schema do not match up. In this situation we still need to transfer data between the two schemas. This data transfer becomes a complexity in its own right. If the in-memory objects know about the relational database structure, then changes in one tend to ripple to the other. The Data Mapper is a layer of software that acts as a mediator between the in-memory objects and the database. Its responsibility is to transfer data between the two, and also the two layers from each other. Using Data Mapper the in-memory objects need have no knowledge that there is even a database present, no SQL interface code, and certainly no knowledge of the database schema.

The separation between domain and data source is the main goal of a Data Mapper, but there are plenty of details that have to be addressed to make it happen. There is also a lot of variety in how different people have built their mapping layers. Figure 2.3 shows the example of Data Mapper (Fowler et al. 2002).

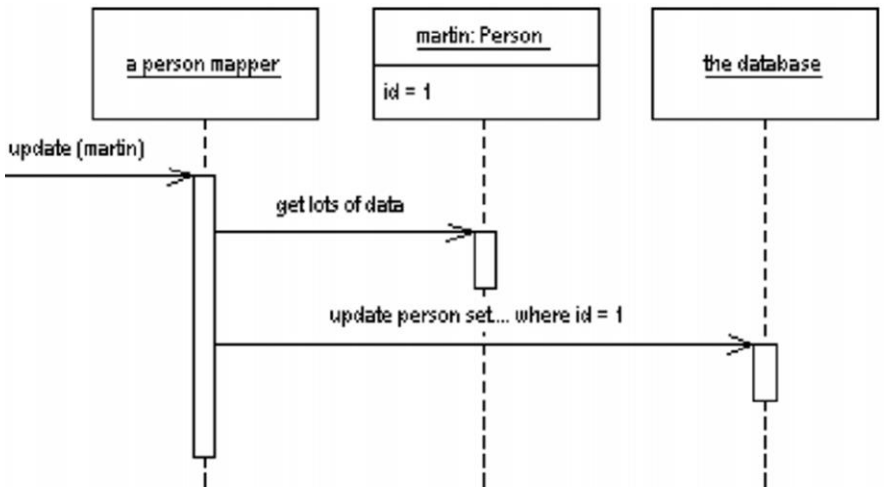


Figure 2.3 Data Mapper Example

Updates data using Data Mapper is simple. A client asks the mapper to save a domain object. The mapper pulls the data out of the domain object and shuttles it to the database. The whole layer of Data Mapper can be substituted, either for testing purposes, or to allow a single domain layer to work with different databases. In this simple case, the mapper separates the database code away from the domain objects, thus making the domain objects simpler as they focus on only one task. But soon other issues come into play which suggest other patterns.

2.2 Software Quality

There are several definitions of software quality. Those definitions are described as follows.

1. There are five perspectives on software quality according to Kitchenham and Pfleeger (Kitchenham & Pfleeger 1996), i.e., (1) transcendental perspective, which is metaphysical aspect of quality; (2)

user perspective, which is the appropriateness of software based on context of use; (3) manufacturing perspective, which is conformance to requirements; (4) product perspective, which is inherent characteristic of the software; (5) final perspective, which is value-based quality.

2. According to Feigenbaum (Feigenbaum 1961), software quality is a customer determination. It is based on customer experiences and measured by their requirements.
3. According to Juran (Juran & Gryna 1988), software quality has several meanings. But there are two categories in common, i.e., (1) quality consist of the product which meet the user requirements and satisfaction; (2) quality consist of the product which free from dependencies.

The International Organization for Standardization (ISO) and the International Electrotechnical Commission (IEC) develop a specialized system for standardization. One of the recent projects is Software product Quality Measurements and Evaluation (SQuaRE). It has five divisions, i.e.:

1. ISO/IEC 2500n – Quality Management Division.
2. ISO/IEC 2501n – Quality Model Division.
3. ISO/IEC 2502n – Quality Measurement Division.
4. ISO/IEC 2503n – Quality Requirement Division.
5. ISO/IEC 2504n – Quality Evaluation Division.

2.2.1 Quality Model Division – ISO/IEC 25010

ISO/IEC 25010 (ISO/IEC 25010 2011) defines two kinds of quality model, i.e., (1) software quality model; (2) system quality in use model. Figure 2.4 shows the scope of quality measures. Data quality model is discussed in ISO/IEC 25012.

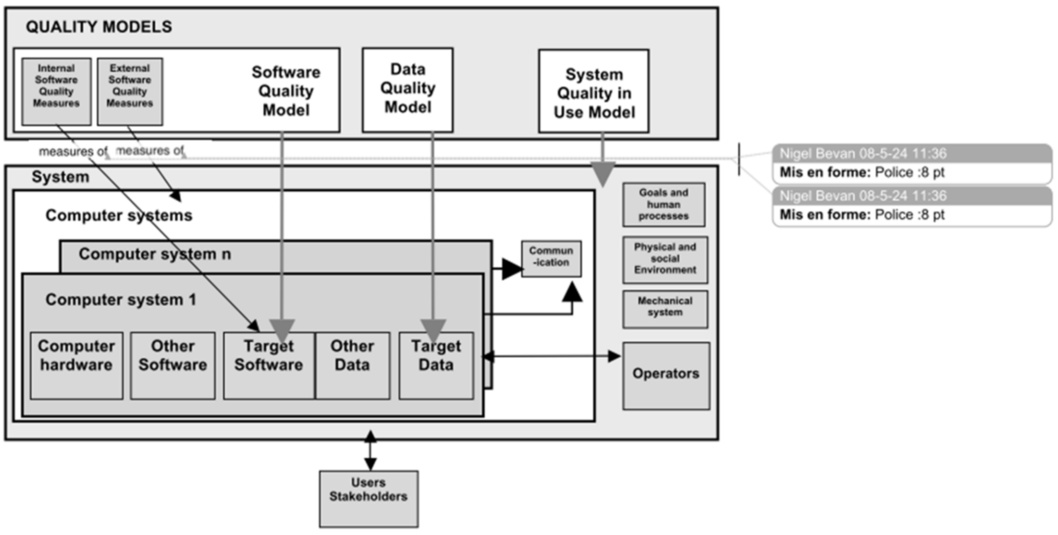


Figure 2.4 Scope of quality measures (ISO/IEC 25010 2011).

Software product quality model composed of eight attributes which can be measured internally or externally. This model can be applied to every kind of software. Those attributes are functional suitability, reliability, performance efficiency, operability, security, compatibility, maintainability, and security. Figure 2.5 shows the characteristic attributes of software product quality model with its sub characteristic.

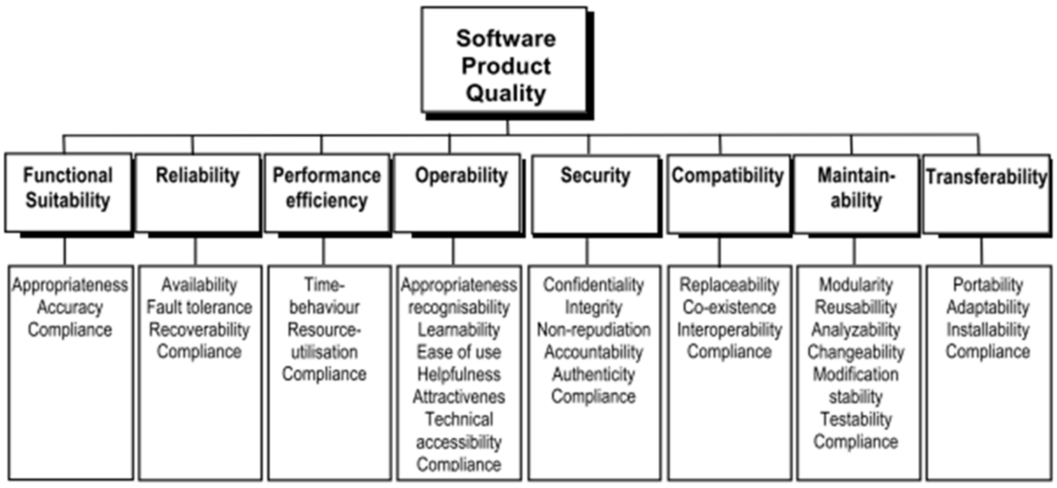

Software Product Quality
Functional Suitability
Reliability
Performance efficiency
Operability
Security
Compatibility
Maintain-ability
Transferability
Appropriateness Accuracy Compliance
Availability Fault tolerance Recoverability Compliance
Time-behaviour Resource-utilisation Compliance
Appropriateness recognisability Learnability Ease of use Helpfulness Attractivenes Technical accessibility Compliance
Confidentiality Integrity Non-repudiation Accountability Authenticity Compliance
Replaceability Co-existence Interoperability Compliance
Modularity Reusabillity Analyzability Changeability Modification stability Testability Compliance
Portability Adaptability Installability Compliance

Figure 2.5 Software product quality model (ISO/IEC 25010 2011).

System quality in use model composed of three characteristic attributes which is presented in Figure 2.6. The product can be measured when it is used in a realistic context. This model may be affected by any of software product quality model attributes. Attributes of this model are usability in use, flexibility in use, and safety in use. Quality in use refers to overall quality in operational environment for a specific user.

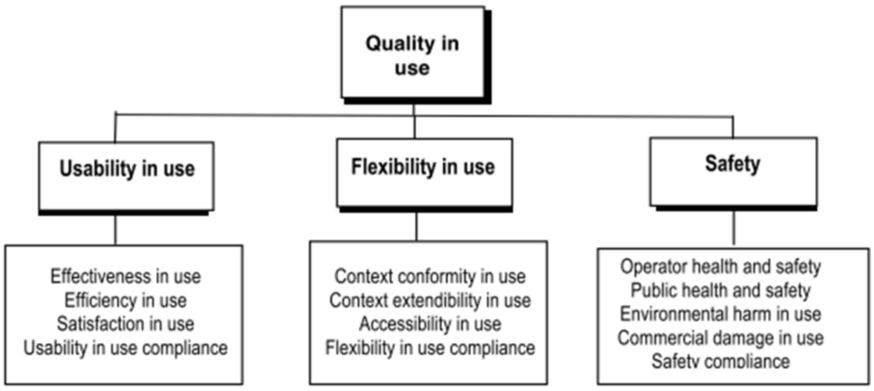


Figure 2.6 System quality in use model (ISO/IEC 25010 2011).

2.2.2 Quality Measurement Division ISO/IEC 25023

ISO/IEC 25023 (ISO/IEC 25023 2015) defines quality measurements function based on characteristic and sub characteristic on ISO/IEC 25010 which is intended to be used together. It contains an explanation of measuring software quality and a basic set of quality measures. The measurable quality-related properties are called properties to quantify. It is measured by applying the measurement method which is a logical sequence of operations. The result of applying the measurement method is called quality measurement element. To produce a quality measure, it is used measurement function to combine the quality measurement element. This quality measure is a quantification of the quality characteristic and sub characteristic. More than one quality measure can be used for the measurement of characteristic and sub characteristic. Figure 2.7 shows the relationship among the measurement.

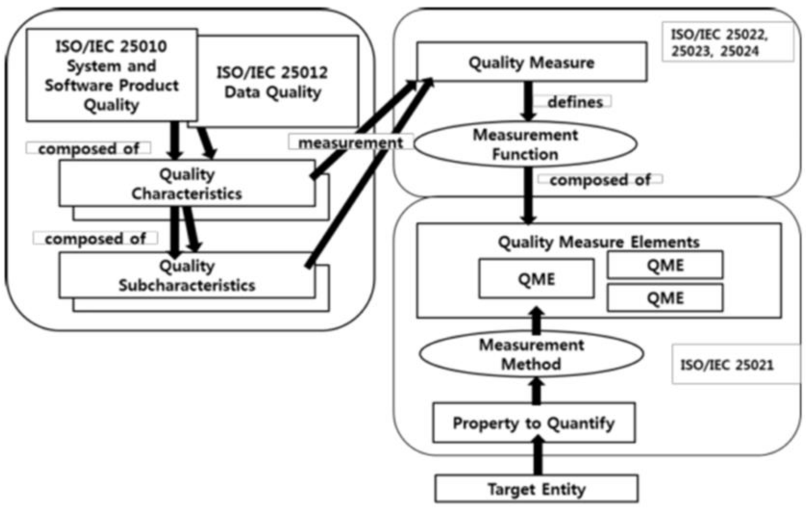


Figure 2.7 Relationship among the measurement (ISO/IEC 25023 2015).

2.2.3 Maintainability

Software maintainability is the degree which the software product can be modified (understood, repaired, and enhanced). Modifications may include corrections, improvement, or adaptation of the software to changes in environment, and in requirements and functional specifications (ISO/IEC 25010 2011). Maintainability has seven sub attributes, i.e.:

- Modularity: The degree to which a system or computer program is composed of discrete components such that a change to one component has minimal impact on other components.
- Reusability: The degree to which an asset can be used in more than one software system, or in building other assets.
- Analyzability: The degree to which the software product can be diagnosed for deficiencies or causes of failures in the software, or for the parts to be modified to be identified.
- Changeability: The degree to which the software product enables a specified modification to be implemented. The ease with which a software product can be modified.

- Modification stability: The degree to which the software product can avoid unexpected effects from modifications of the software.
- Testability: The degree to which the software product enables modified software to be validated.
- Maintainability compliance: The degree to which the software product adheres to standards or conventions relating to maintainability.

There is a relationship between software maintainability and software metrics. Li & Henry (Li & Henry 1993) have validated several object-oriented software metrics. The research found that there is a strong relationship between metrics and maintenance effort in object-oriented software. We can predict maintenance effort by using combination of metrics that are collected from the source code.

2.3 Measurement Methods

This sub chapter discusses the measurement methods. It consists of C&K metrics, ISO/IEC 25023 measurement functions, and Median Absolute Deviation (MAD).

2.3.1 C&K Metrics

This study uses C&K metrics (Chidamber & Kemerer 1994) which have been used widely (Li & Henry 1993; Ampatzoglou et al. 2012). C&K metrics are described as follows.

1. Weighted Methods per Class (WMC)

   Sum of McCabe's cyclomatic complexity of all local methods in the class. Assume a class is C_1 with methods M_1, …, M_n in the class. Let c_1, …, c_n are the complexity of the methods. Then:

$$WMC = \sum_{i=1}^{n} c_i \tag{1}$$

   WMC = n if all method complexities are considered to be unity. Where n is the number of methods.

Viewpoints:

- The number and complexity of methods that involved is become a predictor. It predicts the time and effort is required to develop and maintain the class.
- Large number of methods makes a greater potential impact on children. Children are inheriting all the methods which defined in the class.
- Classes with large numbers of methods are limiting the possibility of reuse.

2. Depth of Inheritance Tree (DIT)

   Inheritance level number of the class, 0 for the root class.

   Viewpoints:

   - The deeper a class is in the hierarchy, the greater the number of methods it is likely to inherit, making it more complex to predict its behavior.
   - Deeper trees constitute greater design complexity, since more methods and classes are involved.
   - The deeper a particular class is in the hierarchy, the greater the potential reuse of inherited methods.

3. Number of Children (NOC)

   Number of direct sub-classes that the class has or number of immediate subclasses subordinated to a class in the class hierarchy.

   Viewpoints:

   - Greater the number of children, greater the reuse, since inheritance is a form of reuse.
   - Greater the number of children, the greater the likelihood of improper abstraction of the parent class. If a class has a large number of children, it may be a case of misuse of sub classing.
   - The number of children gives an idea of the potential influence a class has on the design. If a class has a large number of children, it may require more testing of the methods in that class.

3. Coupling Between Object Classes (CBO)

   Count of the number of other classes to which it is coupled.

   Viewpoints:

- Excessive coupling between object classes is detrimental to modular design and prevents reuse. The more independent a class is, the easier it is to reuse it in another application.
- In order to improve modularity and promote encapsulation, inter-object class couples should be kept to a minimum. The larger the number of couples, the higher the sensitivity to changes in other parts of the design, and therefore maintenance is more difficult.
- A measure of coupling is useful to determine how complex the testing of various parts of a design are likely to be. The higher the inter-object class coupling, the more rigorous the testing needs to be.

4. Response For a Class (RFC)

   Total number of local methods and the number of methods called by local methods in the class. $RFC = |RS|$ where RS is the response set for the class.

   $$RS = \{M\} \cup_{all\ i} \{R_i\} \qquad (2)$$

   Where $\{R_i\}$ = set of methods called by method i and $\{M\}$ = set of all methods in the class.

   Viewpoints:

   - If a large number of methods can be invoked in response to a message, the testing and debugging of the class becomes more complicated since it requires a greater level of understanding required on the part of the tester.
   - The larger the number of methods that can be invoked from a class, the greater the complexity of the class.
   - A worst case value for possible responses will assist in appropriate allocation of testing time.

5. Lack of Cohesion in Methods (LCOM)

   Number of disjoint sets of local methods, i.e., number of sets of local methods that do not interact with each other, in the class. For instance consider a class C with 2 methods M_1 , M_2. Let $\{I_i\}$ = set of instance variables used by method M_i. $\{I_1\} = \{a, b, c, d\}$, $\{I_2\} = \{a, b, c, d, e\}$, then $\{I_1\} \cap \{I_2\}$ is nonempty, which in this case is 1 ($\{e\}$).

   Viewpoints:

- Cohesiveness of methods within a class is desirable, since it promotes encapsulation.
- Lack of cohesion implies classes should probably be split into two or more subclasses.
- Any measure of disparateness of methods helps identify flaws in the design of classes.
- Low cohesion increases complexity, thereby increasing the likelihood of errors during the development process.

Regarding to ISO/IEC 25010 on the maintainability sub attributes, software metrics that used in this study need to be mapped. Each metric represent the complexity of the software. It may affects maintainability in general or the entire sub attributes implicitly. For instance, *WMC* metric is the complexity of class *C*. *WMC* = *n*, where *n* is the number of methods of class *C*. Thus, classes with large number of methods potentially have a bad architecture. It has a greater impact on children and high coupling. It limits the reusability and modularity. Large number of methods is likely harder to trace when there is an error or defect and limits the analyzability. It also becomes a predictor of time and effort that required in maintaining the class which is affects the modifiability (changeability and modification stability). Testing is also more difficult because it harder to predict the behavior of a class with large number of methods.

Mapping is conducted based on which are mentioned explicitly in the literature (Chidamber & Kemerer 1994). Changeability and Modification stability are merged into Modifiability (ISO/IEC 25023 2015). Table 2.2 presents the mapping results between C&K metrics and ISO/IEC 25010 Maintainability sub attributes.

Table 2.2 Mapping between C&K metrics and Maintainability

C&K Metrics	**ISO/IEC 25010 Maintainability sub attributes**
WMC	Modularity, Reusability, Modifiability
DIT	Reusability
NOC	Reusability
CBO	Modularity, Reusability, Modifiability, Testability
RFC	Testability, Modifiability
LCOM	Modifiability

2.3.2 ISO/IEC 25023 Measurement Functions

Based on ISO/IEC 25023, there is also a measurement function that used to measure the maintainability attribute. The maintainability measurement functions which are recommended are stated as follows.

1. Modularity measure
   - ID: MMo-1-G
   - Name: Coupling of components conformance
   - Description: How strongly are the components independent and how many components are free from impacts from changes of other components in a system or software product?
   - Measurement function: X = A / B. A = Number of components which are implemented with minimal impact on others. B = Number of components which required to be independent.
2. Reusability measure
   - ID: MRe-1-G
   - Name: Reusability of assets
   - Description: How many assets in a system can be reusable?

Measurement function: X = A / B. A = Number of assets which are designed and implemented to be reusable B = Number of assets in a system.

2.3.3 Median Absolute Deviation (MAD)

Median Absolute Deviation is also called Absolute Deviation around the Median. It is a robust statistic method to measure central tendency. Robust statistic means it has good performance for a wide ranged and non-normally distributed data. MAD is insensitive to the presence of outliers compared to mean and standard deviation methods. MAD is denoted as (Leys et al. 2013):

$$MAD = b\ M_i\left(\left|x_i - M_j\left(x_j\right)\right|\right) \tag{3}$$

Where,

b = 1.4826 (a constant linked to the assumption of normality of the data),

M = median of the series,

x = population (data).

MAD is used to detect outliers. There are three thresholds depending on the researcher's criteria: 3 (very conservative); 2.5 (moderately conservative); 2 (poorly conservative). Thus the data population which includes for further investigation is:

$$M - threshold * MAD < x < M + threshold * MAD \quad (4)$$

2.3.4 Li & Henry Metrics

This study also uses three additional metrics. These metrics are used by Wei Li and Sallie Henry (Li & Henry 1993) as an addition to C&K metrics in their study to predict maintainability. Table 2.3 shows the additional metrics.

Table 2.3 Additional metrics.

Metric	Description	ISO/IEC 25010 Maintainability sub att.
NOM	Number Of Methods	Modifiability
SIZE1	Lines of code	Modifiability
SIZE2	Number of properties	Modifiability

1. Number of Methods (NOM)

   NOM is a class interface increment metric. It serves well as an interface metric because the local methods in a class constitute the interface increment of the class. It is easy to collect in most object-oriented programming language. The number of local methods define in a class may indicate the operation property of a class. The more methods a class has, it indicates the more complex the interface of the class.

2. Line of code (LOC or SIZE1)

   SIZE1 is one of two size metrics used by Li & Henry. It is used to measure a procedure or function. Then, the accumulated LOC of all procedures and functions is used to measure a program. This metric is measured by counting the number of semicolons in a class.

3. Number of properties (SIZE2)

   SIZE2 is other one of two size metrics. It is calculated by adding the number of attributes and the number of local methods in a class as a number of properties.

2.4 Evaluation Methods

This sub chapter discusses evaluation methods that are used in this study. It consists of Relative Change or Percentage Change and Pearson Product Moment Correlation Coefficient.

2.4.1 Percentage Change

Percentage change or relative change is used in quantitative science to compare two quantities. The term “change” means one of the quantities that being compared is considered as a starting value. For example there are two numerical quantities, x and y, where x as the starting value. Then the relative change is denoted as (Bennett & Briggs 2005):

$$Relative\ change\ (x, y) = \frac{Actual\ change}{x}\% = \frac{\Delta}{x}\% = \frac{y - x}{x}\% \tag{5}$$

The relative change is undefined or zero if the value of x equals zero (0). The value of relative change can be a positive or negative value. Positive value means that the change is increased while negative value means that the change is decreased.

2.4.2 Pearson Product Moment Correlation Coefficient

In statistic, correlation is a measure of linear dependence of two variables or more. This method was found by Karl Pearson in early 90s. The correlation between two variables is not a two way causal relationship. For example the higher the human body, the heavier the body is. However the heavier the human body, does not mean the higher the body is. Thus, there are cause and result in correlation. The Pearson Product Moment Correlation Coefficient (PPMCC), r, can take a range of values from +1 to -1. A value of 0 indicates that there is no association between the two variables. A value indicates a positive association if it is greater than 0. That is, as the value of one variable increases, so does the value of the other variable. A value indicates a negative association if it is less than 0. That is, as the value of one variable increases, the value of the other variable decreases. Pearson product moment coefficient correlation is denoted as (Pearson 1895):

$$r = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \bar{x})^2 \sum_{i=1}^{n}(y_i - \bar{y})^2}} \tag{6}$$

Where:

- r = coefficient correlation
- x_i = *x* value on the *i*th data
- $\bar{x}$ = mean of *x* values
- y_i = *y* value on the *i*th data
- $\bar{y}$ = mean of *y* values

Or it can be denoted as (Pearson 1895):

$$r(x,y) = \frac{\sum xy - \frac{\sum x \sum y}{N}}{\sqrt{\left(\sum x^2 - \frac{(\sum x)^2}{N}\right)\left(\sum y^2 - \frac{(\sum y)^2}{N}\right)}} \tag{7}$$

Where *N* = total number of attributes.

The absolute value of correlation coefficient is used to define the degree of correlation. Table 2.4 shows the degree of correlation.

Table 2.4 Degree of correlation coefficient

r	**Degree of correlation**
0,01 – 0,20	Very low
0,21 – 0,40	Low
0,41 – 0,60	Average
0,61 – 0,80	Strong
0,81 – 0,99	Very strong

2.5 Academic Information System (AIS)

AIS is used to manage the academic data, in this case, data of Institut Teknologi Sepuluh Nopember (ITS). AIS has been used as a case study in several researches (Handani & Rochimah 2015; Rochimah et al. 2015; Rochimah et al. 2014; Yuhana et al. 2016; Sugiyanto et al. 2016). It consists of six modules, i.e., (1) framework; (2) domain; (3) learning; (4) equivalence; (5) curriculum; and (6) assessment (currently in Indonesian language, i.e., framework, domain, *pembelajaran*, *ekuivalensi*, *kurikulum*, and *penilaian*). AIS was developed using Java programming language and Spring MVC for the web development. It also used Eclipse Virgo and OSGI Framework. Tomcat is used for its web server. Figure 2.8 shows package diagram of AIS.

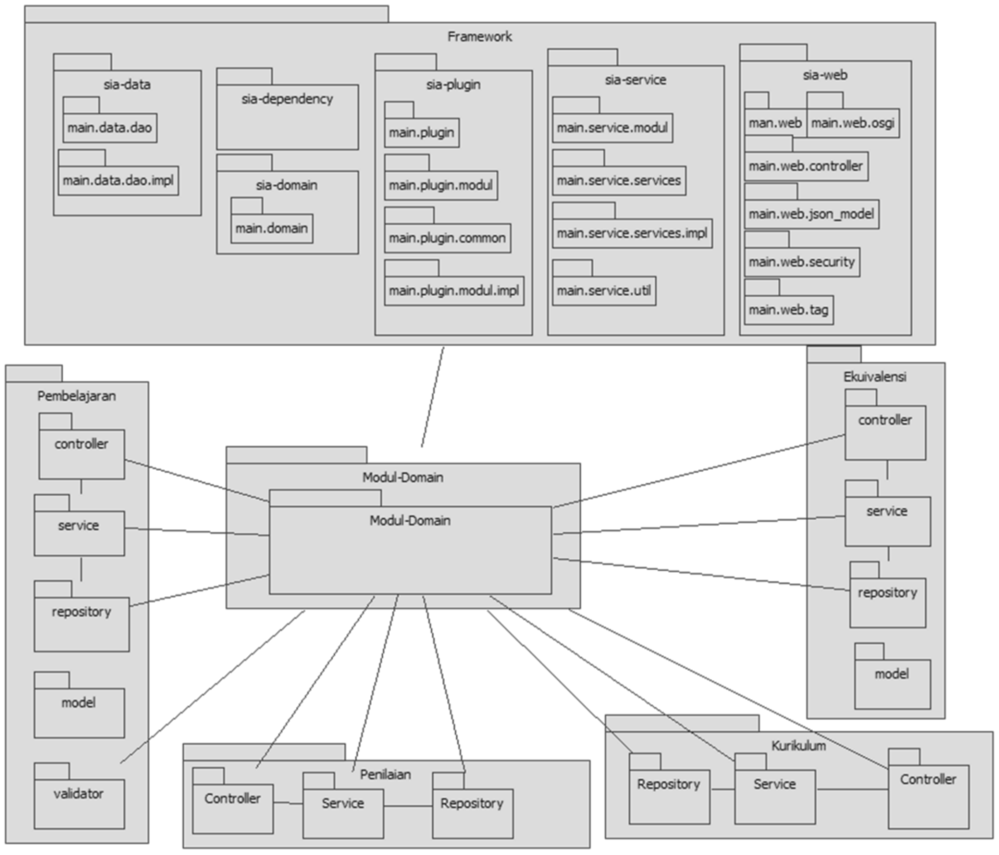


Figure 2.8 AIS package diagram (U. L. Yuhana & Anggraini 2015).

CHAPTER III
RESEARCH METHODOLOGY

3.1 Research Activities

In general, this study begins with literature study, followed by research design and implementation. Writing the report is conducted thoroughly during the research period. Except for reporting, the activities of this study conducted in the following order.

1. Literature study

   This first step is collecting and studying the literatures that related to enterprise software design patterns, refactoring, and software metrics.
2. Research design

   The second step is to construct a research design. It is created based on the result of literature study. It aims to solve the research problem, thus to answer the research questions.
3. Implementation

   The third step is implementing the research design. We follow the research design step by step to produce a result.
4. Reporting

   Every steps of this study is documented, from literature study until implementation.

3.2 Research Design

This study has five main phases, i.e., (1) preparation; (2) measurement of existing system (this is the system which is built without considering the use of design patterns specifically, later it is called as alternative or ALT version); (3) refactoring, (4) measurement of refactored system (this is the system which is built with considering the use of design patterns, later it is called as pattern or PAT version); and (5) evaluation. We present the research design as an activity diagram in Figure 3.1, and the expanded version of each activity in this remaining sub chapter.

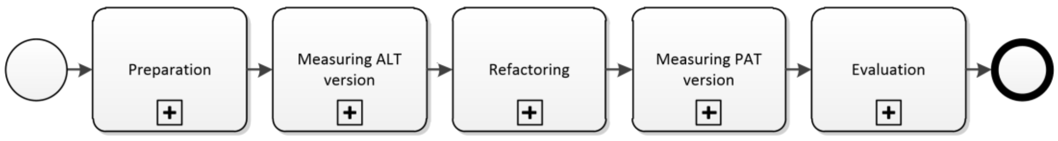


Figure 3.1 Research design activity diagram.

3.2.1 Preparation

Preparation phase includes several definitions, i.e., part of the AIS that used as a case study, quality measurements, and features to be added. This phase, which is shown in Figure 3.2 is a preparation for both ALT and PAT version.

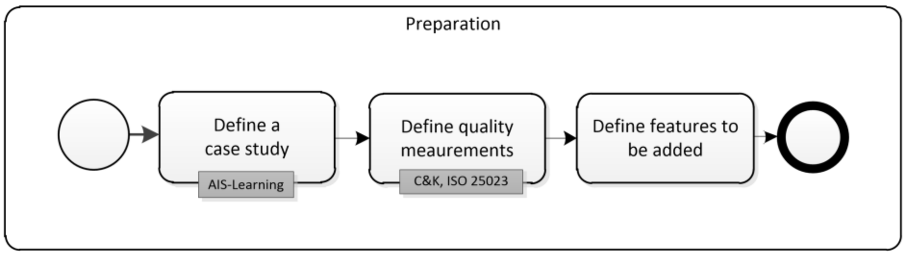


Figure 3.2 Preparation activity diagram.

We use learning module (*pembelajaran*) of AIS as a case study as shown in Figure 3.3. It is often maintained due to changes in standard operating procedure, feature addition or alteration. There are five packages in this module, i.e.:

1. Validator Package

   This package acts as a validator of the incoming data. It also contains several rules which are related to assessment business process.
2. Model Package

   This package is a data mapper between database and program.
3. Repository Package

   This package consists of two kinds of file, i.e., interface and class implementation. It contains Data Access Object (DAO) classes that are used to access the database.
4. Controller Package

   This package represents its name. It contains controller classes that are used to connect presentation and data source layer. It also runs the software business process.

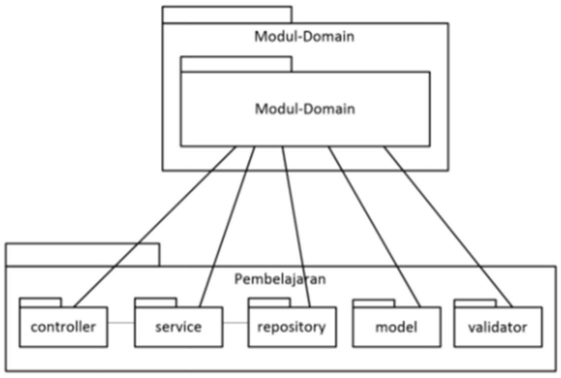


Figure 3.3 Learning module package diagram.

5. Service Package

   This package contains classes that store the AIS business process. Controller call classes from this package to gather data from the repository.

We use C&K metrics (Chidamber & Kemerer 1994) to measure the software maintainability. Those metrics have been used widely (Ampatzoglou et al. 2012; Li & Henry 1993) and can be used to predict maintainability in general. It has been validated with several datasets and techniques. Based on the explanation of each metric, it may represent all of maintainability attributes in ISO/IEC 25010. In addition, we also include two maintainability aspects out of five that are modularity and reusability based on ISO/IEC 25023. Our reason is because those quality aspects are recommended or highly recommended based on its recommendation level. Moreover, it can be measured using the internal software structure.

Features are chosen based on expert judgment. It is because AIS do not have a history of changes. These features are inserted to trigger the source code changes. It has two functions, i.e., (1) as criteria to consider the appropriate pattern; (2) to conduct the experiment of impact analysis on measurement phase.

3.2.2 Measurements

We measure the maintainability of AIS prior to refactoring process (ALT version). Then, we conduct an experiment that is inserting a feature and analyze to what extent the code has changes. Figure 3.4 shows the activity of measuring ALT version. We use the same methods as ALT version to measure PAT version of AIS. Figure 3.5 shows the activity diagram of measurement on PAT version.

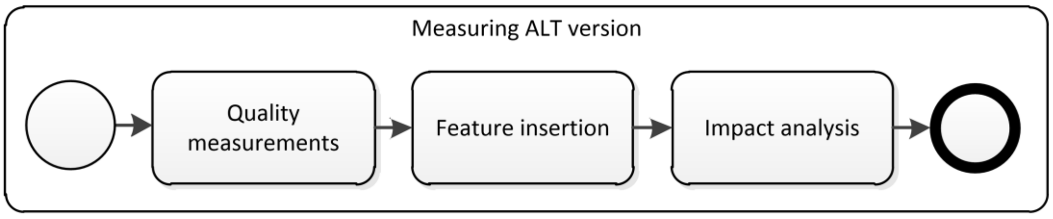


Figure 3.4 Measuring ALT version activity diagram.

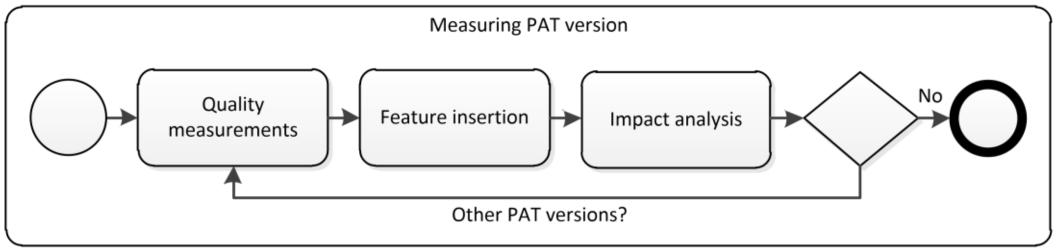


Figure 3.5 Measuring PAT version activity diagram.

There are three different changes that we use in the impact analysis experiment (alteration, addition, and deletion). The quantitative analysis is conducted in two different levels that are class and method level. Consider classes $C_1, \ldots, C_n$ with $M_1, \ldots, M_n$ where M_n is the number of methods in the class C_n. Then, the relative change of the impact change analysis based on equation (5) is stated as follows.

1. Alteration

   Alteration is counted when an existing class or method is altered or modified. It is denoted as:

$$\Delta Alteration_{class} = \frac{C_{altered}}{C_{total}} \times 100\% \tag{8}$$

$$\Delta Alteration_{method} = \frac{M_{altered}}{\sum_{i=1}^{n} M_i} \times 100\% \tag{9}$$

2. Addition Addition is counted when a new class and/or method are added. It is denoted as:

$$\Delta Addition_{class} = \frac{C_{added}}{C_{total}} \times 100\% \quad (10)$$

$$\Delta Addition_{method} = \frac{M_{added}}{\sum_{i=1}^{n} M_i} \times 100\% \quad (11)$$

3. Deletion

   Deletion is counted when a new class and/or method are deleted. It is denoted as:

$$\Delta Deletion_{class} = \frac{C_{deleted}}{C_{total}} \times 100\% \quad (12)$$

$$\Delta Deletion_{method} = \frac{M_{deleted}}{\sum_{i=1}^{n} M_i} \times 100\% \quad (13)$$

3.2.3 Refactoring

At refactoring phase, we develop PAT version and select appropriate patterns based on specific criteria. We simulate to add a feature to trigger the source code changes, thus we analyze and formulate which design pattern are suitable for those case. There is a possibility that more than one pattern are suitable to apply. Thus, there is a possibility we produce more than one PAT version. Figure 3.6 shows the expanded activities of software refactoring.

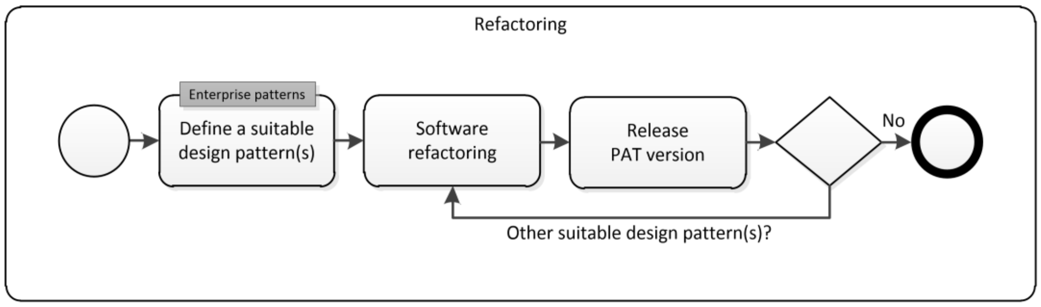


Figure 3.6 Software refactoring activity diagram.

3.2.4 Evaluation

We use Percentage Change or Relative Change which denoted as equation (5) to measure the extent of changes. We separate the measurement between C&K metrics and ISO/IEC 25023 measurement functions because it is measured at a different level. C&K metrics are measured at class level while ISO/IEC 25023 measurement functions are measured at package level. Finally, we calculate the correlation between C&K metrics and ISO/IEC 25023 measurement functions. We use Pearson Product Moment to evaluate the correlation which denoted as equation (6) or (7). We use this method because the maintainability measurement results are in interval scales. Figure 3.7 shows the activity diagram of evaluation phase.

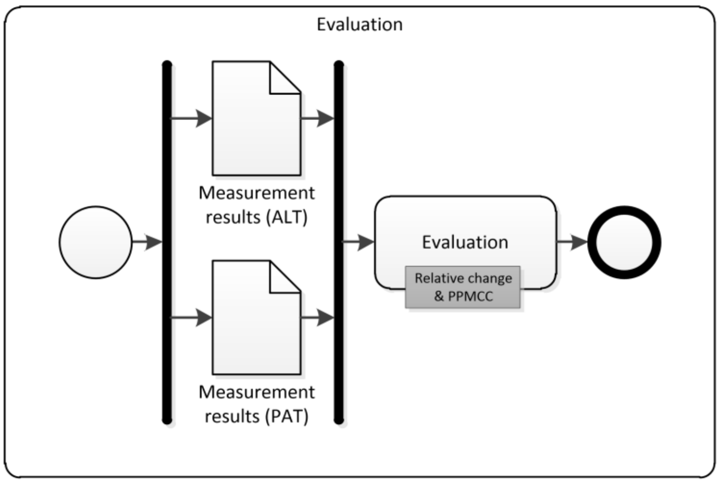


Figure 3.7 Evaluation activity diagram.

3.3 Preliminary Experiment

We conducted preliminary experiment to illustrate how this method is used. This experiment is discussed in the remaining sub chapters.

3.3.1 Preparation

We use a simple Java program as a case study on this experiment, namely Simple Database Manipulation (DbM). It is a CRUD (create, read, update, and delete) program that is used to manipulate database of employee data. Figure 3.8 shows the program’s class diagram.

DbM was built without considering the use of design pattern. So, we call this version as DbM-Alt. It consists of two packages, namely main and presentation. In this case, we ignore the presentation layer and focus on other layer. The main package consists of two classes which are described as follows.

- Employee()

This class consists of domain model and database operations. Domain model represents the Employee data with its attributes. Database operations consist of CRUD. It similar to Active Record design pattern. However, business process on this class is mixed with database operation methods.

- dbConn()

  This class is used to create a database connection object.

DbM-alt manipulates employee data which has four attributes, i.e., (1) id; (2) name; (3) age; (4) salary. Aside from database manipulation, DbM also has a business process. There is a tax that applied 10% of the salary which is stored in a different table with employee id as the foreign key.

We need to define a feature to trigger the source code changes. Both of feature or a change in standard operating procedure is acceptable. In this case, we suppose there is a change in business process. 5% tax is applied for employee under 21 years old and 10% is applied for the others.

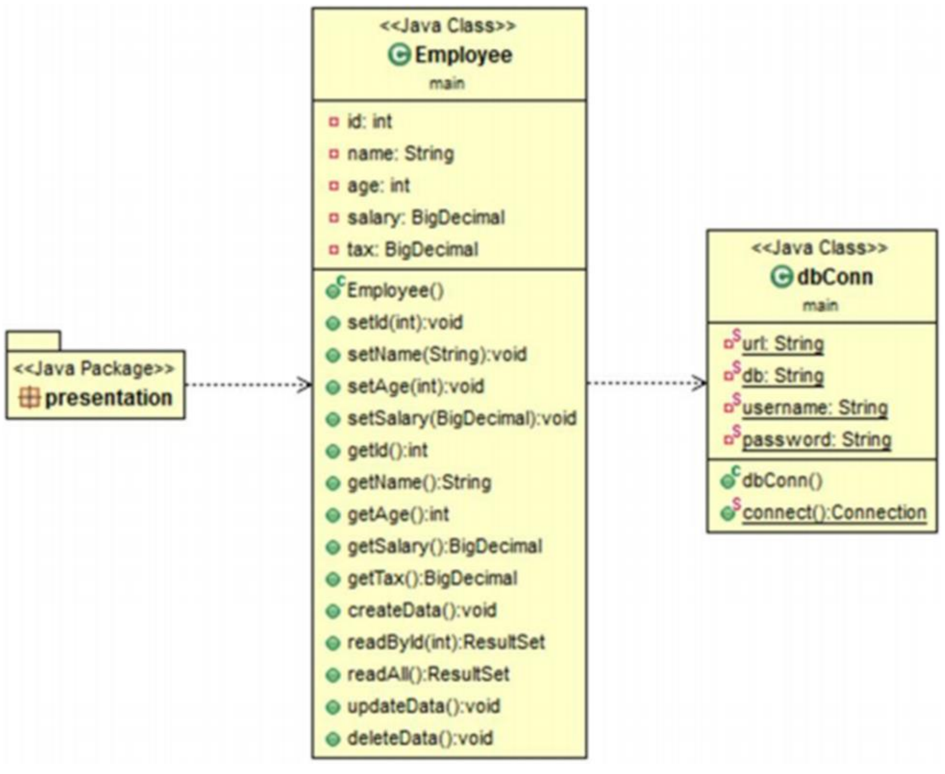


Figure 3.8 DbM-Alt class diagram.

3.3.2 Measuring ALT Version

First step of measurement phase is to measure maintainability by using software metrics. We use Java tool Chidamber and Kemerer Java Metrics (Spinellis 2005) to get the values. For MMo-1-G and MRe-1-G, we measure it manually based on measurement function on ISO/IEC 25023. Terms of component and assets in ISO/IEC 25023 represent a class in this measurement. Thus, we measure it at the package level. Measurement results based on C&K metrics shows in Table 3.1, and based on ISO/IEC 25023 shows in Table 3.2.

Table 3.1 Measurement results of DbM-Alt (C&K metrics).

Classes	WMC	DIT	NOC	CBO	RFC	LCOM
dbConn	3	1	0	0	9	1
Employee	15	1	0	1	30	57

Table 3.2 Measurement results of DbM-Alt (ISO/IEC 25023).

	MMo-1-G	MRe-1-G
main	0.5	0.5

We use MAD to handle outlier data. This step is required to produce a more valid data for evaluation phase because we need to calculate the average or mean of the data. We use 3 as the value of min-max threshold because the data need to be very conservative.

Table 3.3 MAD, min, and max value of DbM-Alt (C&K metrics).

	WMC	DIT	NOC	CBO	RFC	LCOM
MAD	**8.896**	**0**	**0**	**0.74**	**15.567**	**41.513**
Min. Val.	-17.69	1.00	0.00	-1.72	-27.20	-95.54
Max. Val.	35.69	1.00	0.00	2.72	66.20	153.54

Table 3.3 shows the minimum and maximum threshold of DbM-Alt measurement results using C&K metrics. Equation (3) is used to calculate MAD. Equation (4) is used to calculate the minimum and maximum value of the data. It indicates the entire data in Table 3.1 are included for the evaluation phase because there are no outliers. Finally, Table 3.4 presents the average C&K metrics value of DbM-Alt measurement results.

Table 3.4 Average values of DbM-Alt (C&K metrics)

Classes	WMC	DIT	NOC	CBO	RFC	LCOM
dbConn	3	1	0	0	9	1
Employee	15	1	0	1	30	57
Mean	**9**	**1**	**0**	**0.5**	**20**	**29**

For DbM-Alt measurement results using ISO/IEC 25023 measurement functions, it is clearly no need to either calculate MAD or mean because it is a single record of results. We can use those values directly for the evaluation phase.

These values become the starting values to calculate relative changes. It is because there is no classification whether the quality is bad, enough, good, excellent, and so on.

Second step is to simulate the change in standard operating procedure as mentioned at the previous sub chapter. DbM-Alt does not consider the use of design patterns as mentioned by Fowler. If we apply the change, there are two methods in one class that need to modify because of code duplication on its business process. Thus, we need to apply design pattern to organize the domain logic of this application. Detail of the change that occurs based on equations (8, 9, 10, 11, 12) are:

1. Class level
   $\Delta Alteration_{class} = 50\%$
   $\Delta Addition_{class} = 0\%$
   $\Delta Deletion_{class} = 0\%$
2. Method level
   $\Delta Alteration_{method} = 11.8\%$
   $\Delta Addition_{method} = 0\%$
   $\Delta Deletion_{method} = 0\%$

3.3.3 Refactoring

First step of software refactoring is to define a suitable design pattern. We define which design pattern is used based on the code changes in the previous step. As mentioned earlier, there are two methods that need to change. It has a

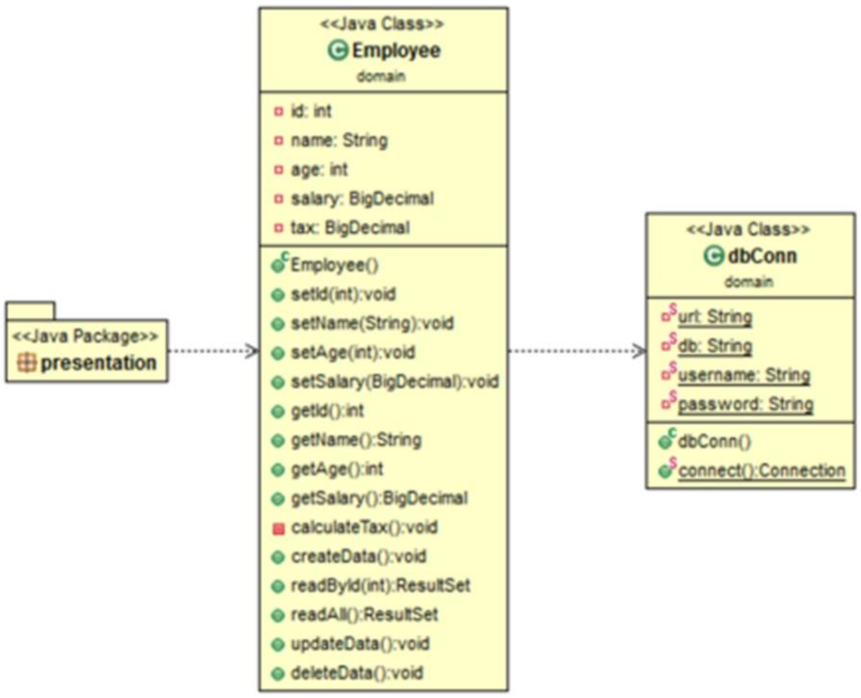


Figure 3.9 DbM-Pat-AR class diagram.

same mechanism to calculate tax before the database interaction process. That means we need to manage the data source architectural patterns. There are four options of design patterns that used to manage the data source, i.e., (1) Active Record Pattern; (2) Table Data Gateway Pattern; (3) Row Data Gateway Pattern; (4) Data Mapper Pattern. We can use all of those design patterns to produce PAT version of DbM. In this experiment, we use domain model to organize the domain logic because considering the future of ever-changing business. Thus, we use Active Record and Data Mapper patterns on this case.

Active Record is addressed to handle a small and simple application. We create a model based on the database, in this case is employee model. This model composed by setter and getter, some business logic, and database query. Thus, it can reduce the duplication method as in DbM-Alt. This class is similar to DbMAlt. However, because we consider the use of design patterns, we pull out the business process becomes a method on this class. Later, this version is called as DbM-Pat-AR. Figure 3.9 shows the class diagram of DbM-Pat-AR.

Data Mapper aims to separate the business logic and database access. It moves the data between domain object and database to keep them independent.

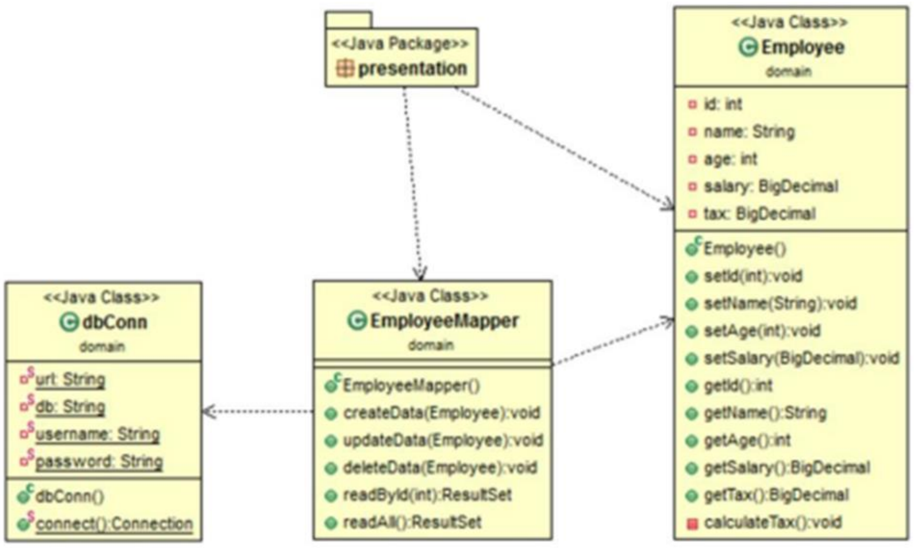


Figure 3.10 DbM-Pat-DM class diagram.

This refactored version is called DbM-Pat-DM. Figure 3.10 shows the class diagram of DbM-Pat-DM.

3.3.4 Measuring PAT Version

We use same measurement methods as we used earlier while measuring DbM-Alt. Table 3.5 and Table 3.6 shows the measurement results of DbM-Pat-AR.

Table 3.5 Measurement results of DbM-Pat-AR (C&K metrics).

Classes	WMC	DIT	NOC	CBO	RFC	LCOM
dbConn	3	1	0	0	9	1
Employee	15	1	0	1	30	53

Table 3.6 Measurement results of DbM-Pat-AR (ISO/IEC 25023).

Packages	MMo-1-G	MRe-1-G
domain	0.5	0.5

To handle the outliers, we use a same mechanism which has been discussed in sub chapter 3.3.2. Table 3.7 presents the minimum and maximum threshold of DbM-Pat-AR measurement results. It shows that the entire data in Table 3.5 is included for evaluation phase. Table 3.8 shows the average values of DbM-PatAR measurement result using C&K metrics. Same thing goes with Table 3.6 as explained in sub chapter 3.3.2.

Table 3.7 MAD, min, and max value of DbM-Pat-AR (C&K metrics).

	WMC	DIT	NOC	CBO	RFC	LCOM
MAD	**8.896**	**0**	**0**	**0.74**	**15.567**	**38.548**
Min. Val.	-17.69	1.00	0.00	-1.72	-27.20	-88.64
Max. Val.	35.69	1.00	0.00	2.72	66.20	142.64

Table 3.8 Average values of DbM-Pat-AR (C&K metrics).

Classes	WMC	DIT	NOC	CBO	RFC	LCOM
dbConn	3	1	0	0	9	1
Employee	15	1	0	1	30	53
Mean	**9**	**1**	**0**	**0.5**	**20**	**27**

Table 3.9 and Table 3.10 show the measurement results of DbM-Pat-DM. Table 3.11 shows the minimum and maximum threshold of DbM-Pat-AR measurement results.

Table 3.9 Measurement results of DbM-Pat-DM (C&K metrics).

Classes	WMC	DIT	NOC	CBO	RFC	LCOM
dbConn	3	1	0	0	9	1
Employee	11	1	0	0	15	41
EmployeeMapper	6	1	0	2	23	15

Table 3.10 Measurement results of DbM-Pat-DM (ISO/IEC 25023).

Packages	MMo-1-G	MRe-1-G
domain	1	0.667

Table 3.11 MAD, min, and max value of DbM-Pat-DM (C&K metrics).

	WMC	DIT	NOC	CBO	RFC	LCOM
MAD	**4.4**	**0**	**0**	**0**	**8.896**	**20.756**
Min. Val.	-7.34	1.00	0.00	0.00	-11.69	-47.27
Max. Val.	19.34	1.00	0.00	0.00	41.69	77.27

In DbM-Pat-DM, there is one value which is considered as outlier that is CBO on EmployeeMapper class. It is an outlier because of the value greater than max threshold. Thus, we calculate the average by using only two of the remaining values as shown in Table 3.12.

Table 3.12 Average values of DbM-Pat-DM (C&K metrics).

Classes	WMC	DIT	NOC	CBO	RFC	LCOM
dbConn	3	1	0	0	9	1
Employee	11	1	0	0	15	41
EmployeeMapper	6	1	0	**2**	23	15
Mean	**6.667**	**1**	**0**	**0**	**16**	**19**

The change in standard operating procedure or business process simulation only affects one class on one method of DbM-Pat-AR and DbM-Pat-DM respectively. The detail is:

- DbM-Pat-AR (Active Record)
  1. Class level

     $\Delta Alteration_{class} = 50\%$

     $\Delta Addition_{class} = 0\%$

     $\Delta Deletion_{class} = 0\%$
  2. Method level

     $\Delta Alteration_{method} = 5.9\%$

     $\Delta Addition_{method} = 0\%$

     $\Delta Deletion_{method} = 0\%$

- DbM-Pat-AR (Active Record)
  1. Class level

     $\Delta Alteration_{class} = 33.3\%$

     $\Delta Addition_{class} = 0\%$

     $\Delta Deletion_{class} = 0\%$
  2. Method level

     $\Delta Alteration_{method} = 5.3\%$

     $\Delta Addition_{method} = 0\%$

     $\Delta Deletion_{method} = 0\%$

Since DbM-Alt-AR has only one class for its domain layer, all of the changes occurred in Employee class. But it is likely harder to trace the code section that needs to change because business logic and database access are in one class.

DbM-Alt-DM is easier to maintain because the business process and database access have been separated in a different class. Based on the change scenario, we can consider that the change is in business process. Thus, the class that become our target is Employee class since it is composed of domain model and business process.

3.3.5 Evaluation

In this evaluation phase, we need to find to what extent the use of design patterns affect software maintainability attribute. We measure the relative change and whether the change is positive or negative. There is a difference of positive or negative meaning on C&K metrics and ISO/IEC 25023. On C&K metrics, positive change means that the value of metric is increased, so the complexity is increased. Thus, the maintainability is decreased and vice versa. Meanwhile on ISO/IEC 25023, positive change means the value of measurement function is increased. Thus, the maintainability is increased because the larger the value, then the higher the cohesion or the reusability is. Table 3.13 shows the relative change of C&K metrics measurement on DbM-Alt (Alternative), DbM-Pat-AR (Active Record), and DbM-Pat-DM (Data Mapper).

Table 3.13 Relative change of DbM (C&K metrics).

Metrics	Alternative (x)	Active Record (y)	Data Mapper (z)	%Δx-y	%Δx-z	%Δy-z
WMC	9	9	6.667	0%	-25.93%	-25.93%
DIT	1	1	1	0%	0%	0%
NOC	0	0	0	0%	0%	0%
CBO	0.5	0.5	0	0%	-100%	-100%
RFC	19.5	19.5	15.667	0%	-19.66%	-19.66%
LCOM	29	27	19	-6.9%	-34.48%	-29.63%
Mean				**-3.39%**	**-28.25%**	**-25.73%**

We use the average or mean of C&K metrics measurement values from each version (Table 3.4, Table 3.8, and Table 3.12) to compose Table 3.13. There are three changes that need to be analyzed, i.e., (1) from Alternative to Active Record, notated as *%Δx-y*; (2) from Alternative to Data Mapper, notated as *%Δx-z*; (3) from Active Record to Data Mapper, notated as *%Δy-z*.

From Alternative to Active Record, the overall Maintainability is increased by 3.39%. There is only one metric value that decreased which means Modifiability is increased based on Table 2.2. Modifiability (ISO/IEC 25023) is Changeability and Modification Stability attributes in ISO/IEC 25010.

From Alternative to Data Mapper, the biggest improvement of Maintainability is occurred by 28.25%. Those values mean Data Mapper is better than Active Record to improve the overall Maintainability in this case. More specific, there are four metrics value that is increased (WMC, CBO, RFC, and LCOM). That means Modularity, Reusability, Modifiability, and Testability attributes are increased.

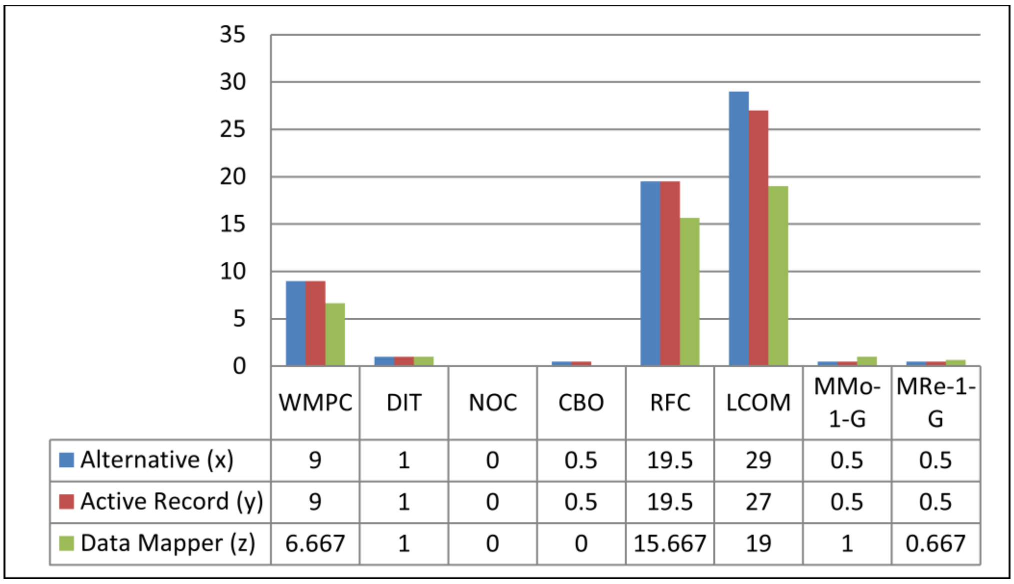
From Active Record to Data Mapper, the overall Maintainability also increased by 25.73%. As mentioned earlier, there is still a room for improvement even we have already used a design patterns. Improvement occurs in four metrics value (WMC, CBO, RFC, and LCOM) which means Modularity, Reusability, Modifiability, and Testability attributes are increased.

Table 3.13 shows the relative change of ISO/IEC 25023 measurement functions on DbM-Alt (Alternative), DbM-Pat-AR (Active Record), and DbMPat-DM (Data Mapper).

Table 3.14 Relative change of DbM (ISO/IEC 25023).

Metrics	Alternative (x)	Active Record (y)	Data Mapper (z)	%Δx-y	%Δx-z	%Δy-z
MMo-1-G	0.5	0.5	1	0%	100%	100%
MRe-1-G	0.5	0.5	0.667	0%	33%	33%
Mean				**0%**	**67%**	**67%**

From Alternative to Data Mapper, there are no improvements either on Modularity or Reusability. From Alternative to Data Mapper, overall Maintainability is increased by 67%. And from Active Record to Data Mapper, overall Maintainability is increased by 67%. Figure 3.11 illustrates the measurement results and Figure 3.12 illustrates the relative change of C&K metrics and ISO/IEC 25023 measurement functions.



	WMPC	DIT	NOC	CBO	RFC	LCOM	MMo-1-G	MRe-1-G
🟦 Alternative (x)	9	1	0	0.5	19.5	29	0.5	0.5
🟥 Active Record (y)	9	1	0	0.5	19.5	27	0.5	0.5
🟩 Data Mapper (z)	6.667	1	0	0	15.667	19	1	0.667

Figure 3.11 Measurement results .

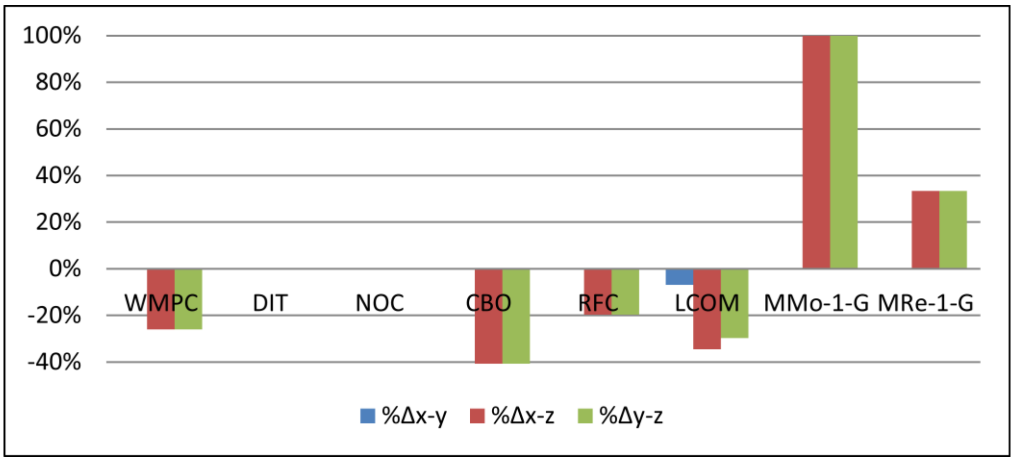


Figure 3.12 Relative changes.

We also calculate the correlation between C&K metrics and ISO/IEC 25023 measurement functions. The result is presented on Table 3.15. Based on those result, C&K metrics are very highly correlated with ISO/IEC 25023 measurement functions. The negative correlation means that the highest the C&K metrics value, then the lowest the ISO/IEC 25023 measurement functions value is. This correlation indicates that the maintainability based on C&K metrics and ISO/IEC 25023 is linear. Which means if the measurement results of C&K metrics found the Maintainability is increased, then the Maintainability of measurement results by using ISO/IEC 25023 is increased too.

Table 3.15 Correlation between C&K metrics and ISO/IEC 25023.

	C&K (a)	**ISO/IEC 25023 (b)**	**a*b**	**a2**	**b2**
%Δx-y	-0.034	0	0	0.001	0
%Δx-z	-0.282	0.667	-0.188	0.080	0.444
%Δy-z	-0.257	0.667	-0.172	0.066	0.444
SUM	**-0.574**	**1.333**	**-0.360**	**0.147**	**0.889**
Correlation coefficient (r)					**-0.996**

[This page is intentionally left blank]

CHAPTER IV
RESULTS & ANALYSIS

4.1 Case Study

Each module of AIS, except Framework and Domain Module, is basically consisted of three packages, i.e. Controller, Service, and Repository (Figure 2.8). It uses a shared domain model (Modul-Domain or Domain Module). While conducted case study selection, we found those shared domain models are considered as “anemic”. Anemic Domain Model is a domain model which has no behavior but a bunch of setters and getters (Fowler 2003). This domain model is an anti-pattern, the opposite of PoEAA (Pattern of Enterprise Application Architecture) which is proposed by Fowler himself. Anemic domain model on AIS causes code duplications. Each module contains behavior of same domain model. Some of the behaviors (service) are exactly the same. Thus, we consider using this problem to select the case study.

Because there is a change on how we select the case study, we cannot use Learning Module alone as mentioned in the previous chapter. The selection involves four modules, i.e. Domain, Learning, Curriculum, and Equivalence Module. It is conducted by selecting one domain model and then investigates its relationships with other modules. Figure 4.1 shows the selected case study which is focused on MK domain model. MK domain model is a model that represents college course with attributes, setters, and getters. We do not use a special method to decide which domain model to be used because all of them are anemic. We just need to make sure that the selected domain model has a relationship with all other modules or as much as possible. Module names in Figure 4.1 are names used by the programmer. Table 4.1 maps the in-picture module names and the actual module names.

Table 4.1 Module names mapping.

In-picture Module Name	Actual Module Name	Abbreviation
com.AIS.Modul.MataKuliah.*	Curriculum	Cr
com.bustan.siakad.*	Equivalence	Eq

com.its.sia.*	Learning	Ln
com.sia.modul.*	Domain	Dm

MK domain model is associated with three modules that are Curriculum, Equivalence, and Learning Module. Each module consists of Controller, Service, and Repository package.

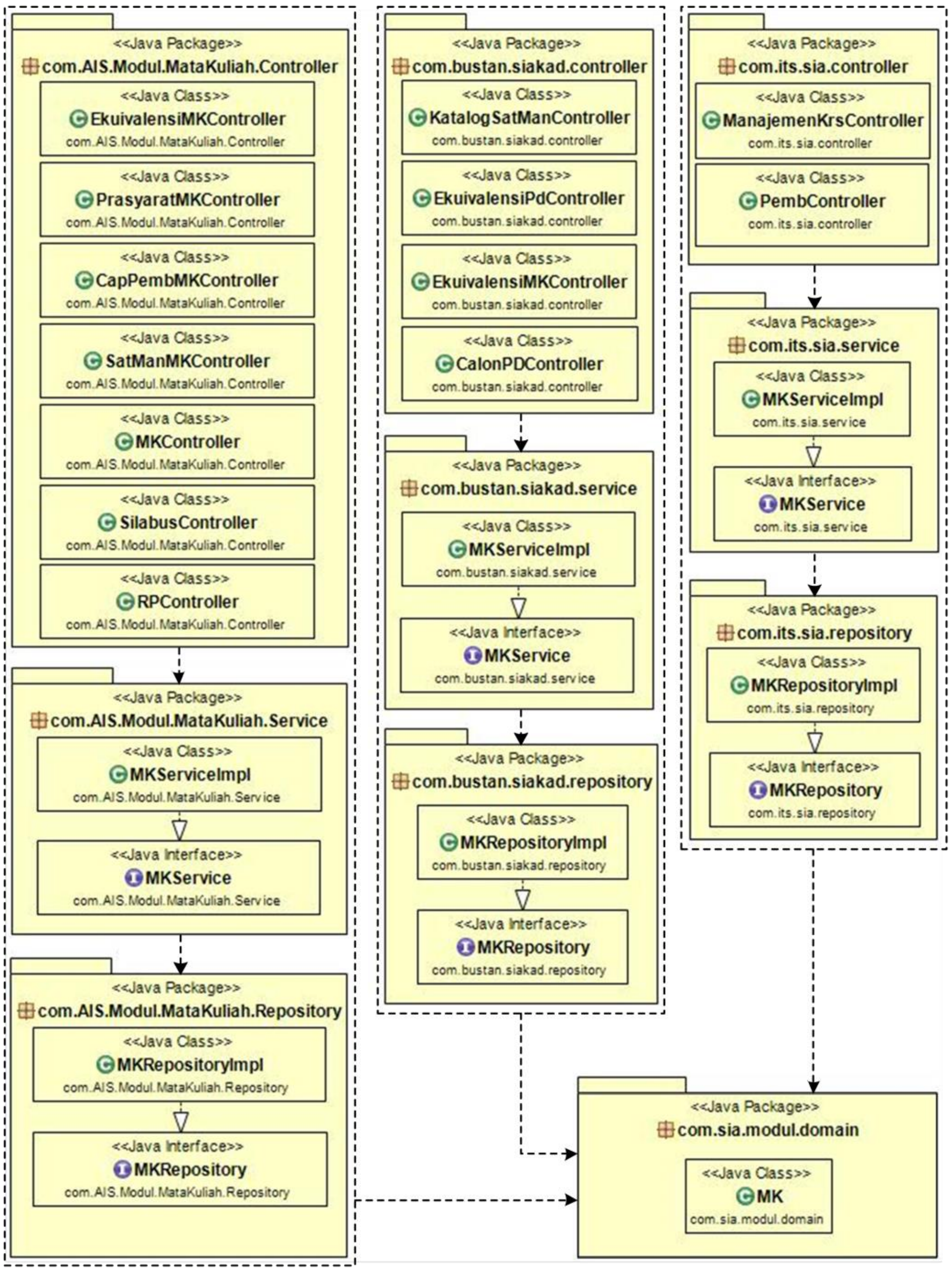


Figure 4.1 Class diagram of case study.

There are duplicated codes on service and repository layer on those three modules. Basically, the service layer on each module is a business process and the repository layer is a data transaction of MK anemic domain model. Thus, they consist of the same code. Figure 4.2 shows the service and repository layer of MK in each module.

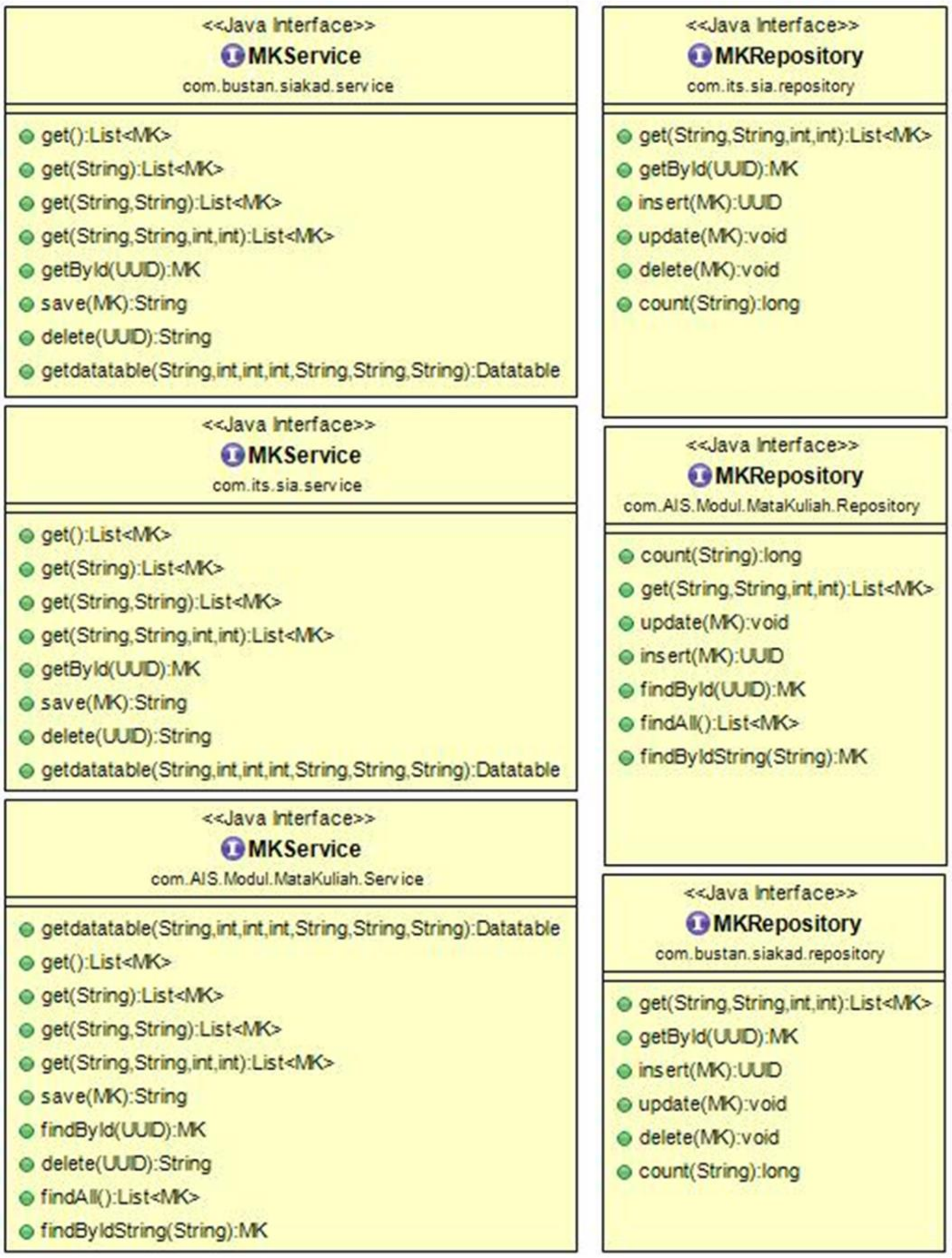


Figure 4.2 Service and repository layer of MK domain model.

Service layer of MK in Equivalence and Learning Module is identically similar. They are also similar with Curriculum Module with one extra method and have a same method with different name (findById and getById). The same thing occurs in repository layer where each layer of those three modules is similar. In maintenance process, if there are changes in business process of MK domain, then all of those Service and Repository classes will change.

Thus, instead of defining and applying features as mentioned in the previous chapter, we consider observing these duplicated methods. Moreover, the duplicated methods is there because the existence of anemic domain model. It becomes the largest impact of those models.

4.2 Refactoring

We develop PAT version by using Domain Logic and Data Source Architectural Patterns because there is a problem with domain model in the current version of AIS. Data Source Architectural Patterns are used to map the domain model into database.

In Domain Logic Patterns, we use Domain Model Pattern because AIS is already using domain model although it is still anemic. In Data Source Architectural Patterns, we pick two patterns which is Active Record and Data Mapper pattern because those patterns suit well with Domain Model Pattern. So, there are two combinations of design pattern which produce two PAT versions of AIS. The first is Domain Model and Active Record Pattern, and the second one is Domain Model and Data Mapper Pattern.

4.2.1 Domain Model and Active Record Pattern

Domain Model is an object model that contains both data and behavior. While Active Record is an object that represents a row in a database table or view and also contain domain logic. Thus, the domain model class of this PAT version will contain data, behavior, and data access. We called this version as PAT-AR version. Figure 4.3 shows the displacement flow of business logic and repository from each module into domain module.

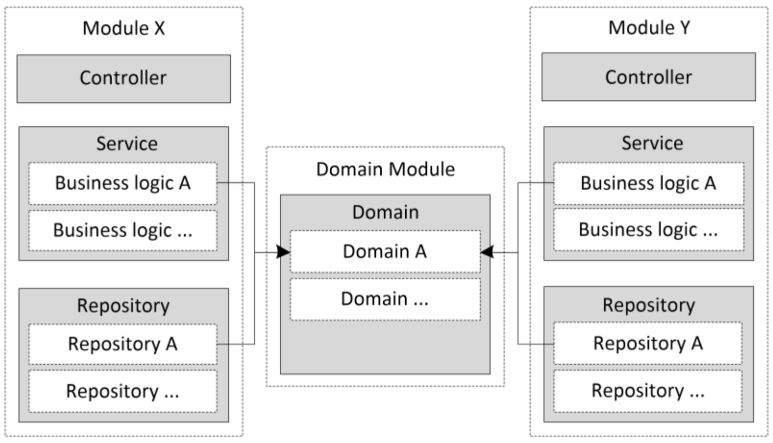


Figure 4.3 Displacement flow of business logic and repository (PAT-AR).

Business logic A from module X and Y merged with its anemic domain in Domain Module. The same goes with repository A from module X and Y also merged with domain A in Domain Module. By this process, now we have domain A which is contains data, behavior, and data access. It also eliminates class duplications in Service and Repository layer. Figure 4.4 shows the architecture of refactored AIS which is PAT-AR version.

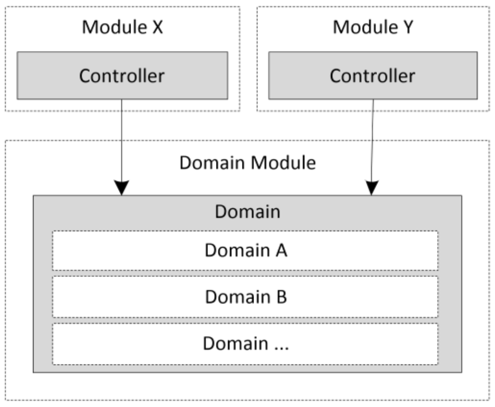


Figure 4.4 Architecture of PAT-AR version.

Based on the displacement flow diagram above, Service and Repository layer from all three modules of AIS (Curriculum, Equivalence, and Learning Module) will be merged into its domain model in Domain Module. Figure 4.5 shows the class diagram of refactored AIS (PAT-AR version).

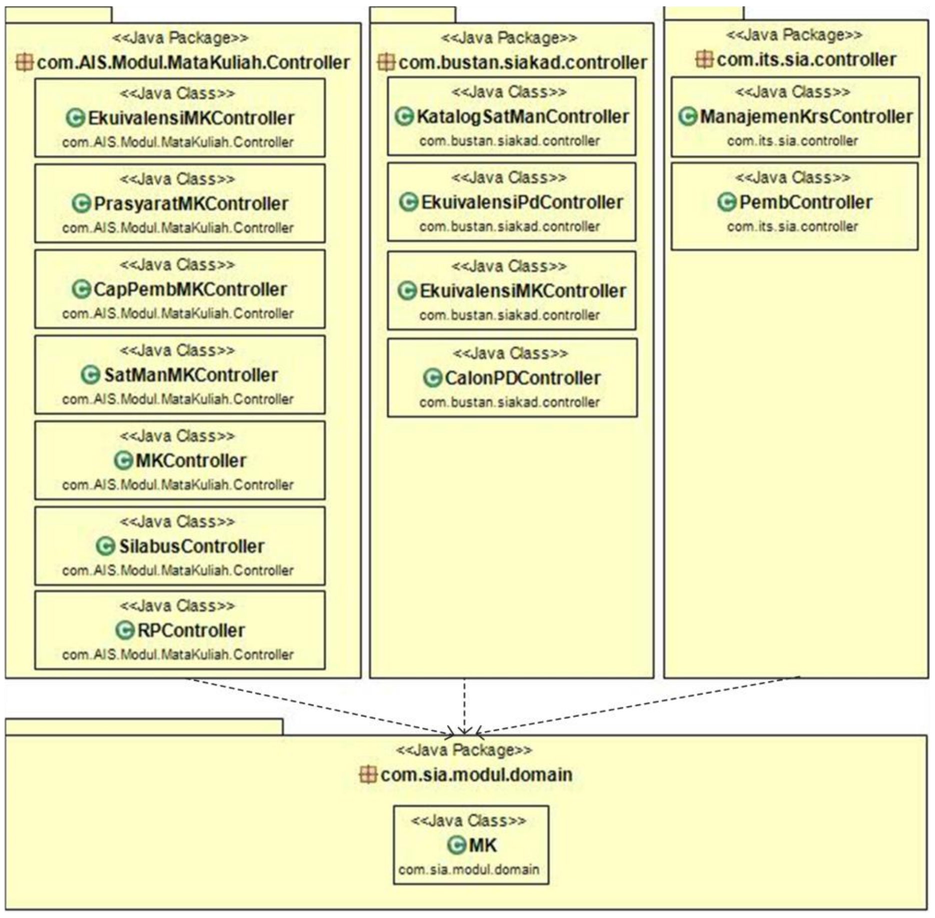


Figure 4.5 Class diagram of PAR-AR version.

4.2.2 Domain Model and Data Mapper Pattern

This version uses the same pattern to manage domain object which is Domain Model Pattern. To manage the data source, this version uses Data Mapper Pattern. Data Mapper is a mapper that moves data between object and database. Thus, the domain object of this version will contain data and behavior while its data access layer is separated from them. We called this version as PAT-DM version. Figure 4.6 shows the displacement flow diagram of this version.

Business logic of Domain A from service layer in other modules is merged into Domain A in domain module. The same thing happens with Domain B, and so on. Thus, that makes the domain model is no longer anemic because it contains both data and behavior.

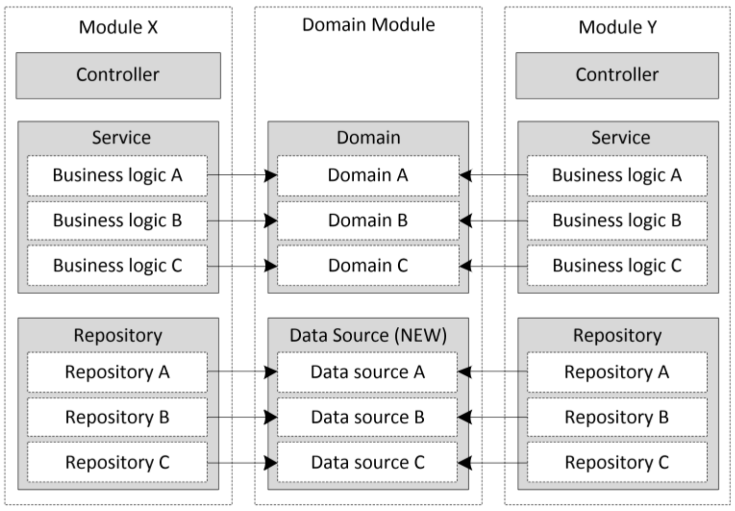


Figure 4.6 Displacement flow of business logic and repository (PAT-DM).

However, there are also duplicated codes in Repository layer. To handle this problem, we make a new layer in Domain Module that is Data-source layer which hold database transaction of domain model. Service and Repository layer in each module (Domain Module excluded) still can contain domain logic and database transaction. If the module uses a unique logic which only applied on that module, it can inherit the related domain model. The same goes with Repository layer. It can inherit the related data-source from Domain Module. The result of this process is shown in Figure 4.7.

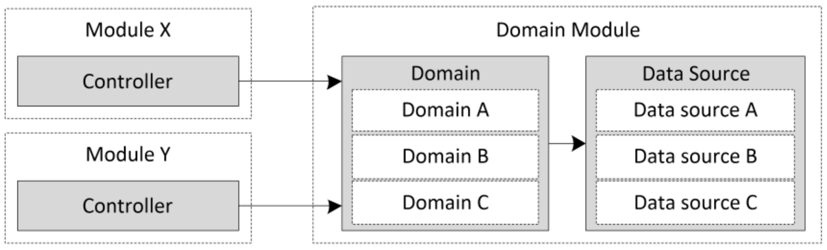


Figure 4.7 Architecture of PAT-DM version.

Service layer from all three modules (Curriculum, Equivalence, and Learning Module) are merged into its domain model in Domain Module. Then, Repository layer from those modules are merged into a new Data-source layer in Domain Module. Figure 4.8 shows the class diagram of refactored AIS (PAT-DM version).

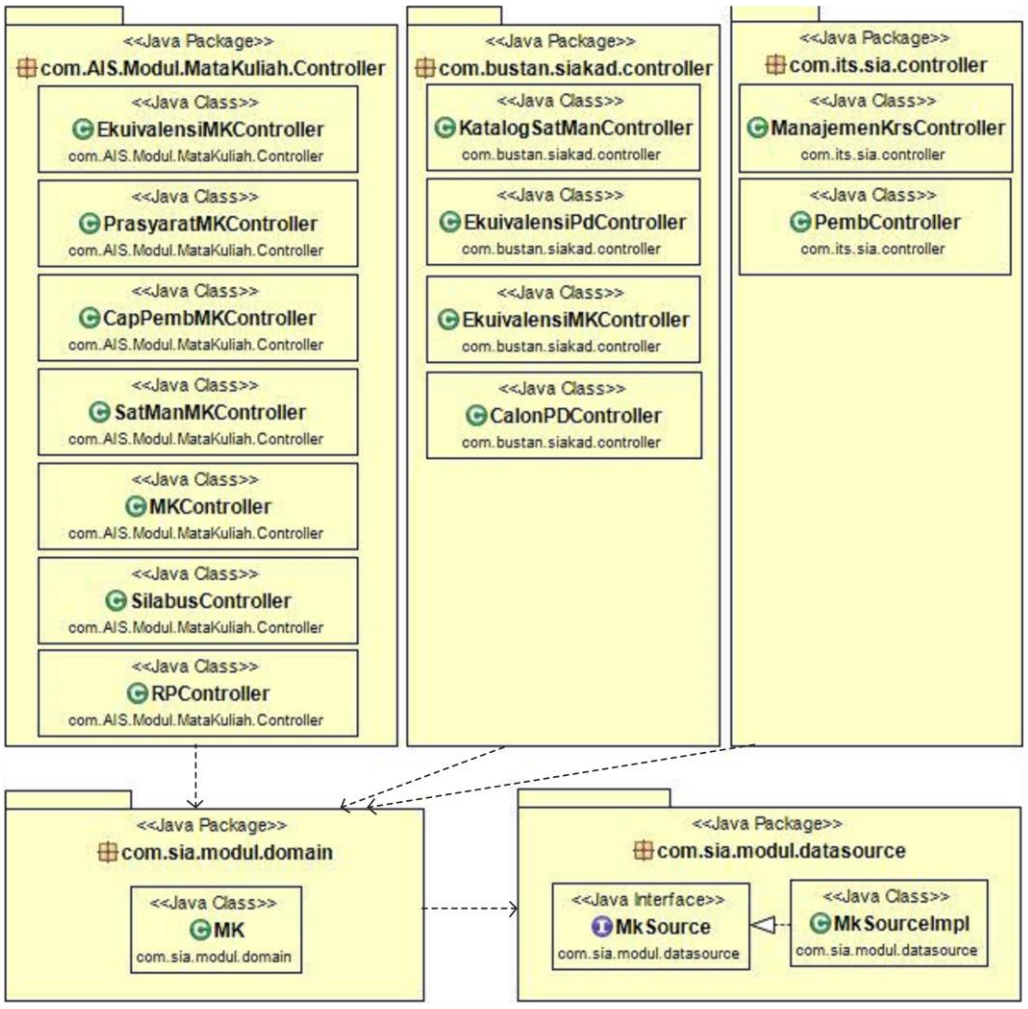


Figure 4.8 Class diagram of PAT-DM version.

4.3 Measurements

Because there is a change in our method to select the case study, we also need to change the measurement method to fit the selected case study. There are three points on this change.

First one is we cannot use the measurement functions of ISO/IEC 25023 because the measurements require the whole classes in a module. In fact, we did not use the whole classes of Learning Module as proposed but involves three modules instead. Moreover, we did not use the whole classes of all those three modules. We only use classes which are related to a specific domain model. In addition to C&K Metrics, we use three more metrics to improve the measurement results. Those metrics are used by Li & Henry to predict software maintainability (Li & Henry 1993).

The second one is we used to detect outliers by using median absolute deviation. It is because we need to produce a more valid average or mean value which is used to evaluate the impact. However, there is a change about how we select the case study and we found that average value alone cannot be used for evaluation. We only use average value where the specific criteria are met, and then we add two new values which are sum and maximum value. Thus, there is no need to detect the outlier using median absolute deviation.

The third one is impact analysis of AIS. We have proposed the impact analysis by simulating the change in standard operating procedure, feature addition, or feature alteration. However, because we push the anemic domain model problem to the surface, we found that the most suitable way to analyze the impact is by investigate the duplicated code or method. Thus, we calculate how many methods in the case study are considered as a duplicate. The result is in percentage with the equation as follows:

$$Duplicated\ methods = \frac{M_{dup}}{\sum_{i=1}^{n} NOM_i}\% \tag{14}$$

Where M_{dup} = number of duplicated method, n = number of classes in case study, and NOM = number of all methods. The method is considered as a duplicate if it is the same as other methods. The term “same” is not exactly the same as it is written in the code, but if they have a same function then it is considered as a duplicate. For example, MKService class of Learning and Equivalence Module in Figure 4.2 are containing same methods. If there are ten methods and eight of them are distinct from other methods, then the number of duplicated method is $M_{dup} = M_{total} - M_{distinct}$. Where $\mathrm{M}_{\mathrm{total}}$ = number of all methods, $\mathrm{M}_{\mathrm{distinct}}$ = number of distinct methods. Then the duplicated methods M_{dup} is 10 – 8 = 2.

4.3.1 Alternative or Non-pattern Version (ALT)

Measurement results of ALT version of AIS are shown in Table 4.2. It involves four modules as show in Figure 4.1. Classes in three modules (Curriculum, Equivalence, and Learning) are consisted of Presentation (Controller), Service, and Repository layer.

Curriculum Module consists of seven classes of Presentation layer, two classes of Service layer, and two classes of Repository layer. Equivalence Module is consisted of four classes of Presentation layer, two classes of Service layer, and two classes of Repository layer. Learning module is consisted of two classes of Presentation layer, two classes of Service layer, and two classes of Repository layer.

Table 4.2 Measurement results of ALT version.

Mod.	Class	WMC	DIT	NOC	CBO	RFC	LCOM	NOM	SIZE1	SIZE2
Cr	SatManMKController	7	1	0	15	40	7	5	126	9
	MKController	7	1	0	17	45	7	5	142	10
	SilabusController	17	1	0	32	105	38	15	344	26
	EkuivalensiMKController	7	1	0	13	35	7	5	113	8
	CapPembMKController	9	1	0	15	58	20	7	181	11
	PrasyaratMKController	7	1	0	12	37	7	5	118	7
	RPController	21	1	0	37	132	56	20	518	37
	MKService	10	1	0	2	10	45	0	17	0
	MKServiceImpl	11	1	0	8	55	11	10	100	13
	MKRepository	7	1	0	1	7	21	0	13	0
	MKRepositoryImpl	8	1	0	6	29	0	7	95	8
Eq	KatalogSatManController	15	1	0	28	145	31	14	521	25
	CalonPDController	34	1	0	44	226	15	33	1381	47
	EkuivalensiMKController	19	1	0	29	160	87	18	836	29
	EkuivalensiPDController	19	1	0	40	196	67	18	1028	32
	MKService	8	1	0	2	8	28	0	14	0
	MKServiceImpl	9	1	0	5	43	14	8	85	11
	MKRepository	6	1	0	1	6	15	0	12	0
	MKRepositoryImpl	4	1	0	6	29	0	6	82	7
Ln	PembController	22	1	0	28	113	113	20	429	31
	ManajemenKRSController	18	1	0	49	147	0	16	473	36
	MKService	8	1	0	2	8	28	0	14	0
	MKServiceImpl	9	1	0	10	57	14	8	100	12
	MKRepository	6	1	0	1	6	15	0	12	0
	MKRepositoryImpl	7	1	0	6	29	0	6	82	7
Dm	MK	23	1	0	3	24	231	22	123	33
Sum		**318**	**26**	**0**	**412**	**1750**	**877**	**248**	**6959**	**399**
Mean		**12.23**	**1**	**0**	**15.85**	**67.31**	**33.73**	**9.538**	**267.7**	**15.35**
Std. Dev.		**7.122**	**0**	**0**	**14.65**	**62.73**	**48.02**	**8.391**	**341.2**	**13.84**
Maximum		**34**	**1**	**0**	**49**	**226**	**231**	**33**	**1381**	**47**

Each layer have a different characteristic, thus standard deviation in some metrics are nearly equal or bigger than its mean because the data are not normally distributed. So, we consider to separating the measurement results based on the layer for evaluation purposes.

The duplicated methods of this version can be investigated by analyzing the class diagram in Figure 4.2. In service layer, total number of method is 26 with 10 distinct methods. Then, the number of duplicated method in this layer is 16 methods. In repository layer, total number of method is 19 with 7 distinct methods. Then, the number of duplicated method in this layer is 12 methods. Total number of duplicated methods in this version is 16 + 12 = 28 methods. Total number of method in this version is 248 methods. By using Equation 14, then:

$$Duplicated\ methods = \frac{28}{248} = 11.29\%$$

4.3.2 Domain Model and Active Record Version (PAT-AR)

Table 4.3 shows the measurement results of PAT-AR version of AIS. This version has 14 classes in total which is less than the number of classes in ALT version (26 classes). Based on the discussion in refactoring phase, the duplicated methods are merged based on its layer and function. Moreover, service layer is merged into domain model to "cure" the anemic model of the domain. In addition, repository layer also merged with domain model to follow the Active Record Patterns. Thus, the number of classes in this version is decreased.

Table 4.3 Measurement results of PAT-AR version.

Mod.	Class	WMC	DIT	NOC	CBO	RFC	LCOM	NOM	SIZE1	SIZE2
Cr	EkuivalensiMKController	7	1	0	11	34	9	5	105	7
	SatManMKController	7	1	0	14	40	7	5	124	9
	SilabusController	17	1	0	30	105	38	15	338	26
	CapPembMKController	9	1	0	14	58	20	7	176	11
	PrasayaratMKController	7	1	0	12	38	7	5	116	7
	MKController	7	1	0	16	45	7	5	138	10
	RPController	21	1	0	35	132	56	20	507	36
Eq	EkuivalensiPDController	19	1	0	39	196	67	18	1000	32
	KatalogSatManController	15	1	0	28	143	31	14	518	25
	EkuivalensiMKController	18	1	0	29	159	83	18	814	29
	CalonPDController	34	1	0	44	225	15	33	1364	47
Ln	ManajemenKRSController	18	1	0	45	148	0	16	444	31
	PembController	22	1	0	28	114	113	20	424	31

Dm	MK	33	1	0	9	72	474	32	255	46
Total		**234**	**14**	**0**	**354**	**1509**	**927**	**213**	**6323**	**347**
Mean		**16.71**	**1**	**0**	**25.29**	**107.8**	**66.21**	**15.21**	**451.6**	**24.79**
Std. Dev.		**8.697**	**0**	**0**	**12.13**	**59.91**	**117.7**	**9.049**	**362.4**	**13.39**
Maximum		**34**	**1**	**0**	**45**	**225**	**474**	**33**	**1364**	**47**

Three modules that are Curriculum, Equivalence, and Learning of this version are only consisted of presentation layer. Curriculum Module has seven controller classes, Equivalence Module has four controller classes, and Learning Module has two controller classes.

As in the previous version, the data on this PAT-AR version of AIS are not normally distributed. For example, standard deviation of LCOM metric is far larger than its mean.

There are no duplicated methods in this version. It is because Service and Repository layer, areas in which those duplicated methods have a high probability to occur, have already merged with domain model. Moreover, this version is using design patterns where the domain object is not anemic anymore.

4.3.3 Domain Model and Data Mapper Version (PAT-DM)

Table 4.4 shows the measurement results of PAT-DM version of AIS. This version has 16 classes in total which is less than the number of classes in ALT version (26 classes) and has two more classes compared with PAT-AR version. Those two classes belong to Data-source layer which is pulled out from domain model to follow the Data Mapper Pattern.

Table 4.4 Measurement results of PAT-DM version.

Mod.	**Class**	**WMC**	**DIT**	**NOC**	**CBO**	**RFC**	**LCOM**	**NOM**	**SIZE1**	**SIZE2**
Cr	EkuivalensiMKController	7	1	0	11	34	9	5	105	7
	SatManMKController	7	1	0	14	40	7	5	124	9
	SilabusController	17	1	0	30	105	38	15	338	26
	CapPembMKController	9	1	0	14	58	20	7	176	11
	PrasayaratMKController	7	1	0	12	38	7	5	116	7
	MKController	7	1	0	16	45	7	5	138	10
	RPController	21	1	0	35	132	56	20	507	36
Eq	EkuivalensiPDController	19	1	0	39	196	67	18	1000	32
	KatalogSatManController	15	1	0	28	143	31	14	518	25
	EkuivalensiMKController	18	1	0	29	159	83	18	814	29
	CalonPDController	34	1	0	44	225	15	33	1364	47
Ln	ManajemenKRSController	18	1	0	45	148	0	16	444	31

	PembController	22	1	0	28	114	113	20	424	31
Dm	MK	32	1	0	6	61	442	31	199	45
Ds	MKSource	7	1	0	1	7	21	0	13	0
	MKSourceImpl	8	1	0	6	30	0	7	90	8
Total		**248**	**16**	**0**	**358**	**1535**	**916**	**219**	**6370**	**354**
Mean		**15.5**	**1**	**0**	**22.38**	**95.94**	**57.25**	**13.69**	**398.1**	**22.13**
Std. Dev.		**8.566**	**0**	**0**	**13.67**	**63.79**	**104.3**	**9.272**	**365.8**	**14.25**
Maximum		**34**	**1**	**0**	**45**	**225**	**442**	**33**	**1364**	**47**

There are no big different of the result summary between this version and PAT-AR version. The overall data of these results are also not normally distributed which is indicated by the value of standard deviation compared to its mean. There are also no duplicated methods in this version.

4.4 Evaluation

This sub-chapter presents the extent of changes that occurs between alternative (ALT), pattern active record (PAT-AR), and pattern data mapper (PAT-DM) version of AIS.

In the previous chapter, section preliminary experiment, we calculated the correlation between C&K Metrics and ISO/IEC 25023 measurement functions. As mentioned earlier, we do not use those measurement functions instead of adding three additional metrics because there is a change in how we select the case study. Thus, we do not calculate the correlation either. We also deepen the analysis in this sub-chapter by discuss the finding which is separated by the logical layers.

First of all, we discuss the relative change of metrics between all those versions of AIS. These changes involve all classes regardless to its layer. We investigate the relative change of three values, i.e. sum, mean, and maximum value of all metrics.

Table 4.5 Relative change of sum value of AIS.

Metric	**Version**			**ΔSum**		
	ALT	**PAT-AR**	**PAT-DM**	**ALT→PAT-AR**	**ALT→PAT-DM**	**PAT-AR→PAT-DM**
WMC	318	234	248	-26.42%	-22.01%	5.98%
DIT	26	14	16	-46.15%	-38.46%	14.29%
NOC	0	0	0	0%	0%	0%
CBO	412	354	358	-14.08%	-13.11%	1.13%
RFC	1750	1509	1535	-13.77%	-12.29%	1.72%
LCOM	877	927	916	5.70%	4.45%	-1.19%
NOM	248	213	219	-14.11%	-11.69%	2.82%
SIZE1	6959	6323	6370	-9.14%	-8.46%	0.74%

SIZE2	399	347	354	-13.03%	-11.28%	2.02%
				-9.72%	**-8.85%**	**0.96%**

Table 4.5 shows the relative change of sum value which is notated by *ΔSum*. From ALT to PAT-AR version, almost all changes of sum value are negative. It indicates a decrease in total complexity of PAT-AR version. Complexity in general, which is represented by metrics, is decreased because the number of classes also decreased. The same goes with the change from ALT to PAT-DM version. The number of classes in PAT-DM version is also less than ALT version. Thus, the total number of metrics is likely to decrease. From PAT-AR to PAT-DM version, there is an increase in the number of classes. Thus, the total number of metrics is likely to increase.

Since PAT-AR and PAT-DM version of AIS has a smaller number of classes compared to ALT version, this result from sum point of view is very reasonable and may or may not represent the general impact of design pattern. We need to break down the evaluation into a smaller scope and investigate the criteria whether the sum value can be used as a valid measurement or not.

Table 4.6 shows the relative change of mean value which is notated by *ΔMean*. From ALT to PAT-AR version, most of the changes are positive. That means the complexity of PAT-AR is greater than ALT version. However, if we investigate the measurement results, each layer has their own nature in term of measurement values. There is also code duplication that makes the evaluation more complicated by using mean value alone. Moreover, some of standard deviation values are bigger than its average value which means the data is not normally distributed. These results also may or may not represent the general impact of design patterns on software maintainability.

Table 4.6 Relative change of average/mean value of AIS.

Metric	**Version**			**ΔMean**		
	ALT	**PAT-AR**	**PAT-DM**	**ALT→PAT-AR**	**ALT→PAT-DM**	**PAT-AR→PAT-DM**
WMC	12.23	16.71	15.50	36.66%	26.73%	-7.26%
DIT	1	1	1	0%	0%	0%
NOC	0	0	0	0%	0%	0%
CBO	15.85	25.29	22.38	59.57%	41.20%	-11.51%
RFC	67.31	107.79	95.94	60.14%	42.54%	-10.99%

LCOM	33.73	66.21	57.25	96.30%	69.73%	-13.54%
NOM	9.54	15.21	13.69	59.50%	43.50%	-10.04%
SIZE1	267.65	451.64	398.13	68.74%	48.75%	-11.85%
SIZE2	15.35	24.79	22.13	61.51%	44.17%	-10.73%
				67.67%	**48.11%**	**-11.66%**

Table 4.7 shows the relative change of maximum value which is notated by *ΔMax*. These values represent the maximum complexity of one class in a module. It is used to indicate the maximum effort required to maintain a class. For example the LCOM value from ALT to PAT-AR version is increased more than doubled. The increased value occurs in Domain Layer which is MK class (see Table 4.2 and Table 4.3). That means the effort which is required to maintain MK class is doubled because MK class in PAT-AR version contains domain logic and data source methods.

Table 4.7 Relative change of maximum value of AIS.

Metric	**Version**			**ΔMax**		
	ALT	**PAT-AR**	**PAT-DM**	**ALT→PAT-AR**	**ALT→PAT-DM**	**PAT-AR→PAT-DM**
WMC	34	34	34	0%	0%	0%
DIT	1	1	1	0%	0%	0%
NOC	0	0	0	0%	0%	0%
CBO	49	45	45	-8.16%	-8.16%	0%
RFC	226	225	225	-0.44%	-0.44%	0%
LCOM	231	474	442	105.19%	91.34%	-6.75%
NOM	33	33	33	0%	0%	0%
SIZE1	1381	1364	1364	-1.23%	-1.23%	0%
SIZE2	47	47	47	0%	0%	0%
				11.04%	**9.44%**	**-1.44%**

Since there are some problems, i.e. standard deviation value bigger than its mean, data are not normally distributed, the existence of duplicated code, the difference in the number of classes between layers, and the difference in the nature of measurement results between layer, we need to split the evaluation based on the layer. We use three layers concept which are Presentation, Domain, and Data-source Layer to covers all layers in all version of AIS. We also use different kinds of value to conclude whether the maintainability is increased or decreased. *ΔSum* is used when comparing a layer that has the same class between versions or when there are duplicated codes. If we assume there are a hundred of duplicated classes (App A) with a metric value of each class is 10. Then two refactored classes (App B) based on App A with metric values are 10 and 12 respectively. So, the comparison of mean value between App A and B is 10:11, which means App A is better than App B even though App A is a bunch of duplicated classes that are more difficult to maintain. If there is a change in App A, then all hundred classes need to change. However, in App B, we only need to manage those two classes without other duplicated classes. So, we use sum and maximum value to evaluate the duplicated codes. *ΔMean* is used when comparing a layer that has no duplicated codes and the standard deviation values is less than its mean. *ΔMax* is used to measure the extent of change in the maximum complexity of one class in a module. Thus, we always include *ΔMax* regardless to the conditions.

4.4.1 Presentation Layer

The comparison involves the same class between versions on this layer. Thus, we use *ΔSum* to evaluate the impact of design patterns.

Table 4.8 Mean and standard deviation values of presentation layer.

Metric	**ALT**		**PAT-AR**		**PAT-DM**	
	Mean	**Std. Dev.**	**Mean**	**Std. Dev.**	**Mean**	**Std. Dev.**
WMC	15.54	7.74	15.46	7.71	15.46	7.71
DIT	1	0	1	0	1	0
NOC	0	0	0	0	0	0
CBO	27.62	12.02	26.54	11.68	26.54	11.68
RFC	110.69	61.54	110.54	61.31	110.54	61.31
LCOM	35	34.27	34.85	33.69	34.85	33.69
NOM	13.92	8.05	13.92	8.05	13.92	8.05
SIZE1	477.69	378.02	466.77	371.81	466.77	371.81
SIZE2	23.69	12.77	23.15	12.48	23.15	12.48

Table 4.8 shows the mean and standard deviation values of metrics. There is no standard deviation value which is bigger than its mean. There is also no duplicated code. Thus, we also us*e ΔMean* for this layer.

Table 4.9 Relative change of metrics from ALT to PAT-AR and PAT-DM.

Metric	**Sum**			**Mean**			**Maximum**		
	ALT	**PAT**	**ΔSum**	**ALT**	**PAT**	**ΔMean**	**ALT**	**PAT**	**ΔMax**
WMC	202	201	-0.50%	15.54	15.46	-0.50%	34	34	0%
DIT	13	13	0%	1	1	0%	1	1	0%
NOC	0	0	0%	0	0	0%	0	0	0%

CBO	359	345	-3.90%	27.62	26.54	-3.90%	49	45	-8.16%
RFC	1439	1437	-0.14%	110.69	110.54	-0.14%	226	225	-0.44%
LCOM	455	453	-0.44%	35	34.85	-0.44%	113	113	0%
NOM	181	181	0%	13.92	13.92	0%	33	33	0%
SIZE1	6210	6068	-2.29%	477.69	466.77	-2.29%	1381	1364	-1.23%
SIZE2	308	301	-2.27%	23.69	23.15	-2.27%	47	47	0%
			-1.83%			**-1.83%**			**-1.17%**

Measurement results of PAT-AR and PAT-DM are the same on this layer. PAT in Table 4.9 represents both of those pattern versions. From sum point of view, the total number of metric is decreased by 1.83% in average. The same goes from mean point of view which is also decreased by 1.83%. It is clear to conclude that the complexity from ALT to PAT version is decreased. The maximum value of metric is decreased by 1.17% in average. The decreased values occur in CBO, RFC, and SIZE1 metric.

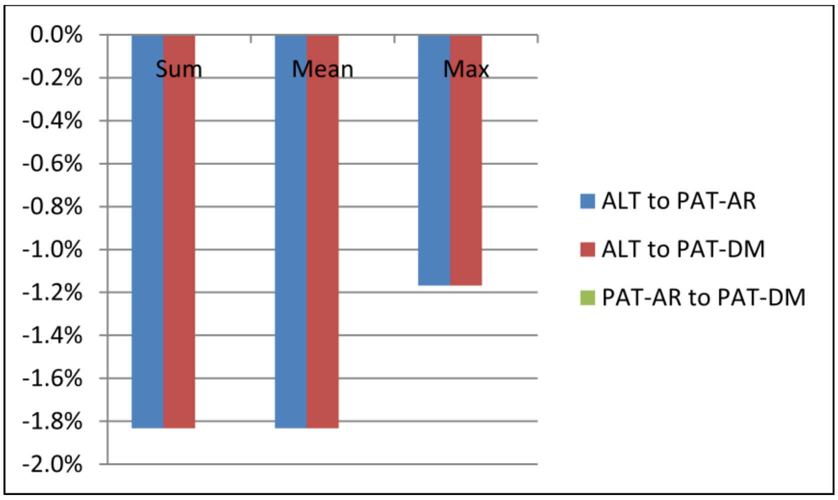


Figure 4.9 Relative change of metrics between versions on presentation layer.

Figure 4.9 shows the graph of relative change between versions in this layer. The bar from PAR-AR to PAT-DM is invincible because the measurement results of both version is the same, thus the relative change is equal to 0%. Negative value of relative change means the complexity is decreased, so the maintainability is increased. Because the measurement results of both pattern versions are the same and the relative change is decreased respectively, both of them have the same impact in improving the maintainability.

Modularity of both pattern versions is increased. It is indicated by the decreased value of WMC and CBO metric. Both of the metric values are decreased by 0.5% and 3.9% in *ΔSum* and *ΔMean* respectively. The maximum value of WMC is unchanged because the methods in this layer remain the same as ALT version. The maximum value of CBO is decreased by 8.16% because in ALT version, presentation layer is connected with a domain model and several services. However, in PAT version, presentation layer only connects with domain model alone.

Reusability of both pattern versions is increased. It is indicated by the decreased value of WMC and CBO metric. The value of DIT and NOC are unchanged. WMC and CBO are decreased by no more than 4%. Moreover, two other metrics remain the same. Thus, the reusability is only increased slightly.

Modifiability of both pattern versions is increased. It is indicated by the decreased value of WMC, CBO, RFC, LCOM, SIZE1, and SIZE2 metric. The value of NOM is unchanged because the methods in this layer are also unchanged. The maximum value of RFC is decreased because the number of methods called by local methods in the class of this layer is decreased. It only connects with one domain model without services from other modules. The maximum value of SIZE1 is decreased because there is a change in how the class of this layer interacts with other modules. Thus, it cuts several lines that contain a code to connect with service layer.

Testability of both pattern versions is increased. It is indicated by the decreased value of CBO and RFC metric by 3.9% and 0.14% respectively. The maximum value of both metrics is also decreased by 8.16% and 0.44% respectively.

As this layer does not have any duplicated methods, both of the pattern versions have no impact related to them. However, pattern versions are able to improve the maintainability to a small extent. The improvement is small because there is not much change that occurs in this layer. Some of the sum and mean values does not change. Any decreased value is also not more than 4%. Moreover, most of the maximum value does not change. In average, the decreased complexity is only by 1.83% from ALT to any pattern versions.

4.4.2 Domain Layer

The comparison of this layer does not involve the same class, however there are duplicated methods. Thus, we use *ΔSum* to evaluate the impact of design patterns.

Table 4.10 Mean and standard deviation of ALT version on domain layer.

ALT	WMC	DIT	NOC	CBO	RFC	LCOM	NOM	SIZE1	SIZE2
Mean	11.14	1	0	4.57	29.29	53.00	6.86	64.71	9.86
Std. Dev.	4.94	0	0	3.02	20.44	73.49	7.40	44.27	10.97

Table 4.10 shows the mean and standard deviation values of ALT version in this layer. Some of standard deviation values are bigger than its mean. Thus, we do not use *ΔMean* to evaluate the impact on this layer.

Table 4.11 Relative change of metrics from ALT to PAT-AR on domain layer.

Metric	Sum			Mean			Maximum		
	ALT	PAT-AR	ΔSum	ALT	PAT-AR	ΔMean	ALT	PAT-AR	ΔMax
WMC	78	33	-57.69%	11.14	33	196.15%	23	33	43.48%
DIT	7	1	-85.71%	1	1	0%	1	1	0%
NOC	0	0	0%	0	0	0%	0	0	0%
CBO	32	9	-71.88%	4.57	9	96.88%	10	9	-10%
RFC	205	72	-64.88%	29.29	72	145.85%	57	72	26.32%
LCOM	371	474	27.76%	53	474	794.34%	231	474	105.19%
NOM	48	32	-33.33%	6.86	32	366.67%	22	32	45.45%
SIZE1	453	255	-43.71%	64.71	255	294.04%	123	255	107.32%
SIZE2	69	46	-33.33%	9.86	46	366.67%	33	46	39.39%
			-27.00%			**411.01%**			**84.40%**

Table 4.11 shows the relative change of metrics from ALT to PAT-AR on this layer. From sum point of view, the total number of metric is decreased by 27% in average. There is one metric value that increased, i.e. LCOM metric. The increased value occurs because we merge the anemic domain, service, and repository into one class. High value of LCOM means classes should probably be split into two or more subclasses. The maximum value of metric is increased by 88.4% in average. It indicates that more effort is needed to maintain the most complex classes in PAT-AR version compared to ALT version. However, there is

only one class that needs to be handled in PAT-AR version. Meanwhile there are seven classes in ALT version. That explains why the sum value is decreased.

Table 4.12 Relative change of metrics from ALT to PAT-DM on domain layer.

Metric	**Sum**			**Mean**			**Maximum**		
	ALT	**PAT-DM**	**ΔSum**	**ALT**	**PAT-DM**	**ΔMean**	**ALT**	**PAT-DM**	**ΔMax**
WMC	78	32	-58.97%	11.14	32	187.18%	23	32	39.13%
DIT	7	1	-85.71%	1	1	0%	1	1	0%
NOC	0	0	0%	0	0	0%	0	0	0%
CBO	32	6	-81.25%	4.57	6	31.25%	10	6	-40%
RFC	205	61	-70.24%	29.29	61	108.29%	57	61	7.02%
LCOM	371	442	19.14%	53	442	733.96%	231	442	91.34%
NOM	48	31	-35.42%	6.86	31	352.08%	22	31	40.91%
SIZE1	453	199	-56.07%	64.71	199	207.51%	123	199	61.79%
SIZE2	69	45	-34.78%	9.86	45	356.52%	33	45	36.36%
			-35.31%			**352.81%**			**63.40%**

Table 4.12 shows the relative change of metrics from ALT to PAT-DM on this layer. The relative change between these versions is similar from the previous comparison. The sum values are decreased by 35.31% with one increased value of metric that is LCOM. In PAT-DM version, we merge anemic domain and service into one class. The maximum value of metric is increased by 63.4% in average. More effort is needed to maintain the most complex classes in PAT-DM version compared to ALT version. However, PAT-DM version is also consisted of one class only.

Table 4.13 Relative change from PAT-AR to PAT-DM on domain layer.

Metric	**Sum**			**Mean**			**Maximum**		
	PAT-AR	**PAT-DM**	**ΔSum**	**PAT-AR**	**PAT-DM**	**ΔMean**	**PAT-AR**	**PAT-DM**	**ΔMax**
WMC	33	32	-3.03%	33	32	-3.03%	33	32	-3.03%
DIT	1	1	0%	1	1	0%	1	1	0%
NOC	0	0	0%	0	0	0%	0	0	0%
CBO	9	6	-33.33%	9	6	-33.33%	9	6	-33.33%
RFC	72	61	-15.28%	72	61	-15.28%	72	61	-15.28%
LCOM	474	442	-6.75%	474	442	-6.75%	474	442	-6.75%
NOM	32	31	-3.13%	32	31	-3.13%	32	31	-3.13%
SIZE1	255	199	-21.96%	255	199	-21.96%	255	199	-21.96%
SIZE2	46	45	-2.17%	46	45	-2.17%	46	45	-2.17%
			-11.39%			**-11.39%**			**-11.39%**

Table 4.13 shows the relative change of metrics from PAT-AR to PAT-DM on this layer. The comparison involves only one class in each version. Thus, all of

those values are the same which is decreased by 11.39%. It indicates that PAT-DM version is better than PAT-AR version in domain layer.

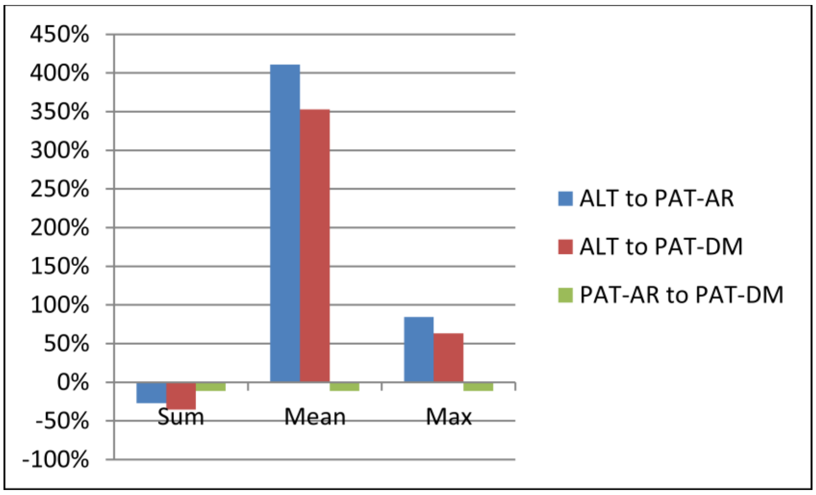


Figure 4.10 Relative change of metrics between versions on domain layer.

Figure 4.10 shows the graph of relative change between versions in this layer. We only use sum and max value, so mean bar is ignored. Maximum bar from ALT to any pattern version is increased. As discussed earlier, the amount of effort needed to maintain the most complex classes in ALT version is less than any of pattern version. However, the amount of effort to maintain the whole classes of ALT version is more than any of pattern version which is indicated by the sum bar. Green-colored bar indicates that PAT-DM version is better than PAT-AR version.

Modularity of both pattern versions is increased. It is indicated by the decreased value of WMC and CBO metric. The sum value is decreased in both of pattern versions. In PAT-AR version, the value is decreased by 57.69% and 71.88% respectively. In PAT-DM version, the value is decreased by 58.97% and 81.25% respectively. The maximum values of WMC are increased in both pattern versions, thus it requires more time and effort to maintain the most complex class. However, pattern versions only consist of one class respectively. So, they still require less time and effort in maintaining their class compared to all classes in ALT version.

Reusability of both pattern versions is increased. It is indicated by the decreased value of WMC, DIT, and CBO metric. NOC metric remains unchanged in pattern versions. It is because there are no changes which involve child classes in all three versions.

Modifiability of both pattern versions is still unclear whether it is increasing or decreasing. Although WMC, CBO, RFC, NOM, SIZE1, and SIZE2 metric values are decreased, there is an increasing value which is LCOM metric. As mentioned earlier, lack of cohesion means the class should probably be split into two or more subclasses. Since we follow the pattern, we cannot split that class. It is not safe to conclude that modifiability is increased just because most of the metric values related to modifiability are decreased. We cannot measure the impact of LCOM metric on other metrics related to modifiability. Thus, another experiment is needed to make the impact more clearly. We discuss this finding further in the next chapter.

Testability of both pattern versions is increased. It is indicated by the decreased value of CBO and RFC metric. In PAT-AR version, the value is decreased by 71.88% and 64.88% respectively. In PAT-DM version, the value is decreased by 81.25% and 70.24% respectively. The maximum value of RFC is increased in both versions by 26.32% and 7.02% respectively. RFC is increased because the total number of methods in a class is greatly increased. However, since any of pattern versions has only one class, the total complexity by RFC metric is still less than ALT version.

In ALT version, this layer consists of 26 methods and 16 of them are duplicates. Thus, 61.54% of the method in this version is duplicates. Any of the pattern versions managed to reduce that value down to zero. Based on case study, pattern versions are able to eliminate the duplicated methods to a great extent regardless of how many they are. On modularity, reusability, and testability sub-attribute, pattern versions are able to improve them to a great extent. The total metric values are decreased by more than 50% of the original complexity. Moreover, there are no more duplicated methods to work with. However, because the modifiability sub-attribute is still unclear, we conclude that the maintainability of pattern versions in this layer is increased to a certain extent.

4.4.3 Data-source Layer

The comparison of this layer does not involve the same class, however there are duplicated methods. Thus, we use *ΔSum* to evaluate the impact of design patterns.

Table 4.14 Mean and standard deviation of ALT version on data-source layer.

ALT	WMC	DIT	NOC	CBO	RFC	LCOM	NOM	SIZE1	SIZE2
Mean	6.33	1	0	3.50	17.67	8.50	3.17	49.33	3.67
Std. Dev.	1.25	0	0	2.50	11.34	8.73	3.18	37.25	3.68

Table 4.14 shows the mean and standard deviation values of ALT version in this layer. Some of standard deviation values are bigger than its mean. Thus, we do not use *ΔMean* to evaluate the impact on this layer.

The comparison of this layer involves only two versions which are ALT and PAT-DM. Technically, PAT-AR version does not have a data-source layer because all of the database transactions are located in domain model.

Table 4.15 Relative change from ALT to PAT-DM on data-source layer.

Metric	Sum			Mean			Maximum		
	ALT	PAT-DM	ΔSum	ALT	PAT-DM	ΔMean	ALT	PAT-DM	ΔMax
WMC	38	15	-60.53%	6.33	7.5	18.42%	8	8	0%
DIT	6	2	-66.67%	1	1	0%	1	1	0%
NOC	0	0	0%	0	0	0%	0	0	0%
CBO	21	7	-66.67%	3.50	3.5	0%	6	6	0%
RFC	106	37	-65.09%	17.67	18.5	4.72%	29	30	3.45%
LCOM	51	21	-58.82%	8.50	10.5	23.53%	21	21	0%
NOM	19	7	-63.16%	3.17	3.5	10.53%	7	7	0%
SIZE1	296	103	-65.20%	49.33	51.5	4.39%	95	90	-5.26%
SIZE2	22	8	-63.64%	3.67	4	9.09%	8	8	0%
			-64.22%			**7.33%**			**-2.29%**

Table 4.15 shows the relative change of metrics from ALT to PAT-DM on this layer. From sum point of view, the total number of metric is decreased by 64.22% in average. Thus, we can conclude that the complexity of PAT-DM version on data-source layer is less than ALT version. It means the maintainability of ALT version on this layer is less than PAT-DM version. The maximum value

of metric is decreased by 2.29% in average. It indicates that less effort is needed to maintain the most complex classes in PAT-DM version compared to ALT version.

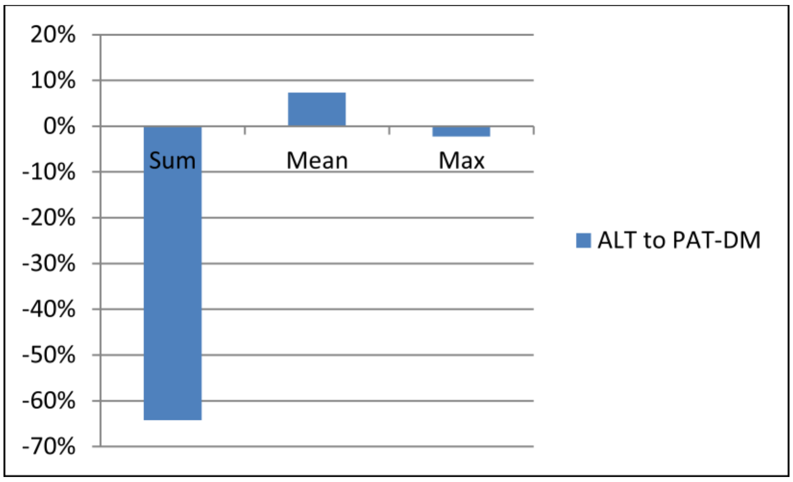


Figure 4.11 Relative change between ALT and PAT-DM on data-source layer.

Figure 4.11 shows the graph of relative change between ALT and PAT-DM version in this layer. We only use sum and max value, so mean bars is ignored. There is one maximum value that is increased, i.e. RFC. However, sum and maximum value itself is decreased in average. PAT-DM version is less complex than ALT version in data-source layer. Thus, PAT-DM has a higher maintainability compared to ALT version.

Modularity of pattern version is increased. It is indicated by the decreased value of WMC and CBO metric. The sum values of those metrics are decreased by 60.53% and 66.67% respectively. There are no changes occur in the maximum value of those metrics.

Reusability of pattern version is increased. It is indicated by the decreased value of WMC, DIT, and CBO metric. NOC value does not change because there are no child classes involved in both ALT and PAT-DM version. Thus, zero percent change does not affect the reusability, unless if the change is positive.

Modifiability of pattern version is increased. It is indicated by the decreased value of WMC, CBO, RFC, LCOM, NOM, SIZE1, and SIZE2 metric. The maximum value of RFC is increased because the class in pattern version contains more methods than ALT version. However, the total number of classes in pattern version is less than ALT version. That explains why the sum value is decreased.

Testability of pattern version is increased. It is indicated by the decreased value of CBO and RFC. Both of the metrics are decreased by 66.67% and 65.09% respectively.

In ALT version, this layer consists of 19 methods and 12 of them are duplicates. Thus, 63.15% of the method in this version is duplicates. Pattern version of this layer is also able to reduce the duplicated methods to a great extent as in domain layer. Pattern version is also able to improve the maintainability to a great extent. It is because the duplicated methods are eliminated. Moreover, the decrease in complexity which represented by the metric values is decreased by more than 50%. It is a great improvement since duplicated methods require more time and effort in doing maintenance.

4.4.4 Threats to Internal Validity

This study uses AIS of ITS as a case study. It contains anemic domain models that cause code duplications in service and repository layer. Without the existence of those duplicated codes, the patterns used may not improve the maintainability to the extent of the results of this study. We may also need other methods to evaluate if there are no duplicated codes in both versions and the standard deviation value is bigger than its mean.

[This page is intentionally left blank]

CHAPTER V
CONCLUSION & FUTURE WORK

5.1 Conclusion

This is a quantitative study to assess the impact of PoEAA on software maintainability. We use AIS of ITS as a case study. AIS is considered as an Anemic Domain Model because the domain model does not contain its behavior. There are five phases which are used in this study, i.e. (1) preparation, (2) measuring non-pattern or alternative version, (3) refactoring, (4) measuring pattern version, (5) evaluation. We use nine software metrics to measure the complexity and to predict the software maintainability. There are three design patterns that are used in this study. We use Domain Model as its Domain Logic Pattern, then Active Record and Data Mapper as its Data Source Architectural Pattern. We use combinations of those three patterns that produce two pattern versions. In the evaluation phase, we calculate the relative change of each metric and evaluate it based on the layers. We compare the measurement results of all versions based on three layers, i.e. presentation, domain, and data-source. The conclusion of this work can be drawn as follows:

1. Pattern selection is conducted based on the layer. The selected case has a problem in its domain layer which is anemic, thus we decide to organize the domain logic by using Domain Logic Patterns. The anemic domain model also affects data-source layer, thus we use Data Source Architectural Patterns to solve the problem.
2. On presentation layer, both of the pattern versions have a same measurement results. They managed to improve the maintainability of ALT version. Modularity, reusability, modifiability, and testability of pattern versions are increased. It is indicated by a decreased metric value. However, the decreased value is no more than 4%. Moreover, there are no duplicated methods in this layer. This concludes that the pattern versions are able to improve the maintainability to a small

extent. Change that occurs in this layer is not too much and the decrease in complexity is only by 1.83% in average.

3. On domain layer, PAT-AR version is able to improve the maintainability of ALT version. The number of duplicated methods is reduced to zero, or decreased by 100% from 61.54% to 0% of duplicated methods. Modularity, reusability, and testability sub-attribute are also increased. The metric values which represent those sub-attributes are decreased by more than 50%. However, modifiability sub-attribute is still unclear because there is one increasing metric value. Therefore, PAT-AR version is only able to improve the maintainability to a certain extent. The same goes for PAT-DM version. The different is PAT-DM version is able to improve the maintainability more than PAT-AR version. It is improved to a certain extent because the total complexity is only decreased by 11.39% and no duplicated methods are involved.
4. On data-source layer, PAT-DM version is able to improve the maintainability of ALT version. Modularity, reusability, modifiability, and testability of PAT-DM version are increased. The decrease in complexity which represented by the metric values is decreased by more than 50%. The number of duplicated methods is also decreased by 100% from 63.15% to 0%. PAT-DM version is able to improve the maintainability to a great extent.
5. The greatest maintainability improvements occur on data-source layer, followed by domain layer, and then presentation layer. That is because data-source layer of ALT version is the least layer which uses design patterns. Domain layer is already using domain model though still anemic and there are no patterns involve in presentation layer.
6. PoEAA can “heal” the anemic domain model of AIS also eliminate the duplicated methods in service and repository layer of ALT version of AIS. The impact can be evaluated by measuring the metric values in each version and comparing them.

7. The duplicated code from ALT to any of pattern version is decreased by 11.29%.

Despite design patterns are able to improve the software maintainability and eliminate code duplications, there are several drawbacks:

1. While duplicated methods are eliminated, the average maximum value of metric in any pattern version is greatly increased.
2. There is lack of cohesion in domain layer of pattern version. The value of LCOM metric is increased, which mean the complexity is increased. However, the increased value is reasonable since domain layer holds both data and behavior instead of setter and getter only. LCOM metric indicates the Modifiability. While this metric is increased, other metrics which also represent Modifiability is decreased. That is why Modifiability in domain layer is still unclear.

5.2 Future Work

In future work, we need to investigate the drawbacks of this work. The sub-attribute of maintainability that remain unclear is Modifiability. We may solve this problem by conducting an experiment which involves volunteers to maintain AIS. The experiment is designed according to the Modifiability sub-attribute. Thus, we can investigate the correlation between the value of software maintainability metrics and the software maintenance activities.

[This page is intentionally left blank]

REFERENCES

Ali, M. & Elsih, M.O., 2013. A Comparative Literature Survey of Design Patterns Impact on Software Quality. In *Proceeding of the International Conference on Information Science and Applications (ICISA)*. pp. 1–7.

Ampatzoglou, A., Charalampidou, S. & Stamelos, I., 2013. Research state of the art on GoF design patterns: A mapping study. *Journal of Systems and Software*, 86(7), pp.1945–1964.

Ampatzoglou, A., Frantzeskou, G. & Stamelos, I., 2012. A methodology to assess the impact of design patterns on software quality. *Information and Software Technology*, 54(4), pp.331–346.

Bennett, J. & Briggs, W., 2005. *Using and Understanding Mathematics: A Quantitative Reasoning Approach (3rd ed.),* Boston: Pearson.

Chidamber, S.R. & Kemerer, C.F., 1994. A Metrics Suite for Object Oriented Design. *IEEE Transactions on Software Engineering*, 20(6), pp.476–493.

Christopoulou, A. et al., 2012. Automated refactoring to the Strategy design pattern. *Information and Software Technology*, 54(11), pp.1202–1214.

Feigenbaum, A.V., 1961. *Total Quality Control*, McGraw-Hill.

Fowler, M., 2003. Anemic Domain Model. Available at: https://martinfowler.com/bliki/AnemicDomainModel.html [Accessed July 6, 2017].

Fowler, M. et al., 2002. *Patterns of Enterprise Application Architecture*, Addison Wesley.

Gamma, E. et al., 1994. *Design Patterns: Elements of Reusable Object-Oriented Software*, Pearson Education.

Gonzalez-Sanchez, J. et al., 2012. Affective computing meets design patterns: A pattern-based model for a multimodal emotion recognition framework. In *EuroPLoP '11 Proceedings of the 16th European Conference on Pattern Languages of Programs*.

Handani, F. & Rochimah, S., 2015. Relationship Between Features Volatility And Software Architecture Design Stability In Object- Oriented Software :

Preliminary Analysis. In *2015 International Conference on Information Technology Systems and Innovation, ICITSI 2015 - Proceeding*. pp. 1–5.

ISO/IEC 25010, 2011. Systems and software engineering - Systems and software Quality Requirements and Evaluation (SQuaRE) - System and software quality models.

ISO/IEC 25023, 2015. Systems and software engineering - Systems and software Quality Requirements and Evaluation (SQuaRE) - Measurement of system and software product quality.

Juran, J.M. & Gryna, F.M., 1988. *Juran's Quality Control Handbook*, McGraw-Hill.

Kitchenham, B. & Pfleeger, S.L., 1996. Software quality: the elusive target. *IEEE Software*, 13(1), pp.12–21.

Leys, C. et al., 2013. Detecting outliers: Do not use standard deviation around the mean, use absolute deviation around the median. *Journal of Experimental Social Psychology*, 49(4), pp.764–766.

Li, W. & Henry, S., 1993. Object-Oriented Metrics that Predict Maintainability. *Journal of Systems and Software*, 23(2), pp.111–122.

Muraki, T. & Saeki, M., 2002. Metrics for Applying GOF Design Patterns in Refactoring Processes. In *Proceedings of the 4th international workshop on Principles of software evolution - IWPSE '01*. p. 27.

Pearson, K., 1895. Notes on regression and inheritance in the case of two parents. In *Proceedings of the Royal Society of London*. pp. 240–242.

Rochimah, S., Rahmani, H.I. & Yuhana, U.L., 2015. Usability characteristic evaluation on administration module of Academic Information System using ISO/IEC 9126 quality model. In *2015 International Seminar on Intelligent Technology and Its Applications, ISITIA 2015 - Proceeding*. pp. 363–368.

Rochimah, S., Yuhana, U.L. & Raharjo, A.B., 2014. Academic Information System Quality Measurement Using Quality Instrument : A Proposed Model. In *2014 International Conference on Data and Software Engineering, ICODSE 2015 - Proceeding*. pp. 1–6.

Spinellis, D., 2005. Tool writing: A forgotten art? *IEEE Software*, 22(4), pp.9–11.

Sugiyanto, Rochimah, S. & Sarwosri, 2016. The improvement of software quality model for academic websites based on multi-perspective approach. *Journal of Theoretical and Applied Information Technology*, 86(3), pp.464–471.

U. L. Yuhana, G.P.N.S. & Anggraini, R.N.E., 2015. *Rancang Bangun Commercial Off The Shelf (Cots) Sistem Informasi Akademik Berbasis Web Pada Modul Kelola Pembelajaran*. Institut Teknologi Sepuluh Nopember, Surabay: JURNAL TEKNIK ITS.

Yuhana, U.L., Saptarini, I. & Rochimah, S., 2016. Portability characteristic evaluation Academic information System assessment module using AIS Quality Instrument. In *ICITACEE 2015 - 2nd International Conference on Information Technology, Computer, and Electrical Engineering: Green Technology Strengthening in Information Technology, Electrical and Computer Engineering Implementation, Proceedings*. pp. 133–137.