Refactoring the Anemic Domain Model using Pattern of Enterprise Application Architecture and its Impact on Maintainability: A Case Study

Siti Rochimah, *Member, IAENG*, I Made B. Gautama, Rizky J. Akbar, *Member, IAENG*

Abstract— **Design pattern is a set of solutions that is used to solve software development common problems. The purpose of design pattern utilization is to improve software quality. Various design patterns have been proposed. One of them is Patterns of Enterprise Application Architecture (PoEAA) which are specified for enterprise application. However, there are lacks of literature that discuss these patterns. This research conducts a quantitative study to assess the impact of design pattern on software maintainability. We use Academic Information System of Institut Teknologi Sepuluh Nopember as a case study. It is an enterprise software which has Anemic Domain Model. We perform refactoring into the existing systems using suitable PoEAA. We measure its maintainability using C&K and three additional metrics, prior and after the refactoring process. The measurement results are then evaluated to obtain the impact. Based on the experiments, we clearly observe that PoEAA utilization could significantly restructure the anemic domain model of AIS. The maintainability is increased especially in presentation layer. PoEAA also eliminates duplicated methods in service and repository layer of the existing version of AIS. However, there are several drawbacks of the improvements.**

Index Terms— **academic information system, design patterns, maintainability, software evolution, software metrics.**

I. Introduction

Design pattern is 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 [1]. Research of software design patterns in various fields is still conducted until nowadays. There are several design patterns that have been proposed, i.e., GoF 1994, Buschmann 1996, Sinha 1996, Fowler 2002, Serial 2011, and so on [2]. Design pattern consists of a set of solution which is used to solve software development common problems. Thus, it shortens software development, reduces costs, and improves the software quality [3],[4]. Usually, design pattern cannot be used directly into the source code because it takes the form of a description or template. It is used to guide the software development to produce a more reusable code.

This research 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 standard operating procedure, as well as features addition or alteration. Maintenance process is difficult because one change in certain code affects other codes in several places. Doing this process repetitively may increase structure complexity of the software. Thus, the future maintenance process will be much more 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 [5]. This research involves the application of design patterns to lead the software refactoring. We use enterprise software design patterns by Fowler [6]. The main reason of utilizing those patterns is because AIS involves persistent data. Applying design patterns aim to improve the software maintainability.

The AIS that is used in this research is the AIS that has been replicated into an experimental environment. This original AIS was first built without considering certain design patterns. It was done without using any standards and not all of the AIS is well managed [7]. Based on the result of the previous research [8], AIS needs to be evolved. AIS structure becomes more complex along with how often the maintenance is conducted. It is one of the challenges in software evolution which is called software erosion [9]. Thus, it needs a re-engineering to fix the problem. The reengineering process utilizes design patterns to improve the software maintainability. There are so many literature that discusses design pattern, which may be used for references. However, the impact of design patterns on software quality attributes are still controversial [3],[10]. There are also lacks of literature that studied enterprise design patterns specifically. Hence, we conduct a quantitative research to study the impact of enterprise design patterns on software maintainability. We measure the software maintainability using software quality metrics [5],[11]. We also investigate

Manuscript received April 13, 2018; revised March 23, 2019. This work was supported by The Directorate of Research and Community Service, Ministry of Research and Higher Education of Indonesia (DRPM – RISTEKDIKTI Indonesia) under the Contract Number: 86/Addendum/ITS/2017.

Siti Rochimah and Rizky J. Akbar are with the Department of Informatics, Institut Teknologi Sepuluh Nopember, Surabaya Indonesia, email: siti@if.its.ac.id, rizky@if.its.ac.id.

I Made B. Gautama is with the Department of Informatics, STIKOM Bali, Indonesia, email: bhaskaragautama@gmail.com.

the duplicated code which is an impact of Anemic Domain Model of AIS. The purpose of this research is to produce scientific evidence in which design patterns may improve the software maintainability. The result of this study is expected to help developer in determining appropriate patterns when conducting reengineering on AIS.

II. Background

A. Patterns of Enterprise Application Architecture

There is no big difference between GoF and enterprise design pattern in definition and purpose. The difference lies in which type of software it is suitable for use. There are also distinct kinds of enterprise software. Thus, enterprise design pattern offers multiple solutions instead of only one. All of the patterns are about choices and alternatives [6].

According to Fowler, there are several factors that indicate a software is an enterprise software.

(1) *Persistent data*. In general, enterprise application usually involves persistent data which are needed to be around among multiple runs of the program and persist for several years. Many changes may occur on this data along with the use of the program. Structural changes may occur to store new pieces of information without disturbing the old data. The data will still persist even if the company decides to use new software to manage their data.

(2) *Organize a lot of data*. Enterprise software usually uses large size of database, mostly relational database. Moderate system has at least over 1 GB of data in ten millions of records [6].

(3) *Many people access data concurrently*. Enterprise software usually used by many people, at least less than a hundred. However, the number of people may increase significantly on web-based software that communicates over the internet.

(4) *A lot of user interfaces screen*. Because of huge data to work with, there is usually a lot of user interfaces screen to handle the data. With many people access the data, they need to be presented in many ways for many different purposes.

(5) *Integrated with other enterprise software*. Enterprise application is usually integrated with other enterprise applications to perform the job. They may be considered as a different system although they are able to communicate with common communication technology.

(6) *Differences in business process and conceptual dissonance with the data.* For example, a term "customer" may be different for several department of a company. One department thinks a customer is someone which has a current agreement. Another department may think a customer is someone who has ever been make an agreement with the company, although not any longer.

(7) *Consist of complex business logic*. Business logic may be 'illogic' since every company has its own unique business process.

B. Software Maintainability

Software maintainability is the degree in which the software product can be easily modified [12], i.e. understood, repaired, and enhanced. Modification may include correction, improvement, or adaptation of the software due to changes in environment, requirement, or in functional specification.

There is a relationship between software maintainability and software metrics. Li & Henry [13] 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. The maintenance effort can be predicted by using software metrics for maintainability measurement purpose.

C. C&K Metrics

C&K metrics is a metric that suites for object oriented design. It has been used to predict software maintainability [13]. It predicts software complexity in general. C&K metrics consist of six metrics which are shown in Table 1.

TABLE 1 C&K METRICS [13]

Metric	Description
WMC	Weighted Methods per Class. Sum of McCabe's cyclomatic complexity of all local methods in the class [11].
DIT	Depth of Inheritance Tree. Inheritance level number of the class, 0 for the root class.
NOC	Number Of Children. Number of direct sub-classes that the class has or number of immediate subclasses subordinated to a class in the class hierarchy.
CBO	Coupling Between Object classes. Count of the number of other classes to which it is coupled.
RFC	Response For a Class. Total number of local methods and the number of methods called by local methods in the class.
LCOM	Lack of Cohesion in Methods. 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.

We also use three additional metrics of Li & Henry to predict software maintainability [13]. Those metrics are presented in Table 2.

TABLE 2 ADDITIONAL METRICS [13]

Metric	Description
NOM	Number Of Methods. The number of local methods defined in a class. Number of methods related to class's complexity.
SIZE1	Lines of code or number of semicolons in a class.
SIZE2	Number of properties. This size metric is number of attributes and number of local methods in a class.

D. Academic Information System

AIS is used to manage academic data, in this case, data of ITS. AIS has been used as a case study in several studies [7], [8],[14],[15]. It consists of six modules: (1) framework; (2) domain; (3) learning; (4) equivalence; (5) curriculum; and (6) assessment. 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 the web server.

The architecture of AIS is considered as Anemic Domain Model which is mentioned by Fowler [16]. It is one of those anti-patterns in object-oriented programming. Based on the three principal layers, i.e. presentation, domain logic, and data source, domain logic in Anemic Domain Model consist of objects without its behavior but setters and getters. This is

what makes the domain model is considered as 'anemic'. Domain model in object-oriented design is supposed to hold both data and behavior or process altogether. Thus, Anemic Domain Model is considered as a procedural style design, so that it is contradictory to the basic idea of object-oriented design.

Four modules of AIS which are learning, equivalence, curriculum, and assessment use a similar structure. It consists of three packages, i.e. controller, service, and repository. Controller package belongs to presentation layer as part of Model-View-Controller (MVC) Pattern. Service package consist of business logic. Repository package consist of database transaction script. They use a shared domain model which is domain module. Domain module consists of domain model for other modules, so that it is called as an Anemic Domain Model. Business logic is scattered around controller and service package in each module. The impact is that the emersion of numerous duplicated codes in service package (laid on service layer) across modules. For instance, if a process of calculating student grades is needed by all of five modules, then each service in each modules will do the same process. Fig. 1 shows the basic structure of those AIS modules.

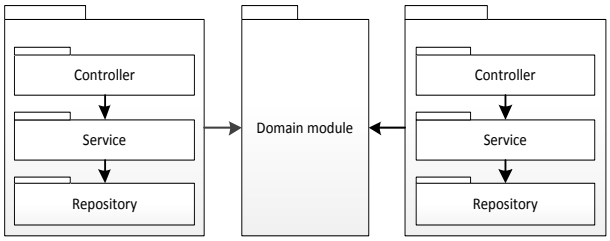

Fig. 1. Basic Architecture of AIS.

III. Related Work

This section discusses studies that have been done related to the impact of design patterns on software maintainability.

In 2011, Ampatzoglou et al. investigated the reusability of design patterns and software packages [17]. Based on ISO/IEC 25010, reusability is one of maintainability sub-attributes [12]. The study uses 100 open source projects with 27.461 classes as case studies. It involves eleven GoF design patterns. They investigated a scenario where the desired requirement is implemented as a design patterns. Classes that should be used as a starting point for white-box reuse is selected in order to optimize the reusability of the selected classes. The investigation is to find out which unit is the most reusable: a class, a pattern, or a package. They found that the alternative to reuse the design pattern offers optimal selection option in most of the cases. Although there are also cases where the package alternative offers a more reusable set of classes.

In the same year, Gonzalez-Sanchez et al. created an object-oriented software model for Intelligent Tutoring System using design patterns. They experienced project development for three years using the proposed model. Several aspects become their focus on the study such as creating a common language among stakeholders, supporting an incremental development, and adjustment to a highly shifting development team. The study used GoF design patterns to create the object-oriented software model. They found that design patterns are useful to create a high-quality software solution which is easy to maintain and extend. It also improves their communication, collaboration, and productivity whithin teamwork. With highly shifting development team, design patterns make the knowledge transfer becomes easier and faster. Design patterns also allow them to create a common vocabulary among stakeholders. The result is that they managed to improve the maintainability by using design patterns.

In 2012, Nanthaamornphong and Carver conducted an experiment to study whether design patterns improve software maintainability and understandability. It involves GoF design patterns and eighteen participants in a graduate-level software engineering course. The experiment of understandability is to create a new application, whilst the experiment of maintainability is to replicate the existing application. As a result, design patterns did not improve either maintainability or understandability. They mentioned that it is not always useful to use design patterns. They also suggested that developer should study the impact of design patterns before using them.

In 2014, Bernardi et al. proposed a framework to improve the implementation of design patterns by using Model Driven Development techniques along with Aspect Oriented Programming (AOP). It involved GoF design patterns such as Command, Composite, Strategy, and Singleton patterns. They used two different implementations of Java system as case studies. One system is implemented using their proposed framework. Whilst the other system is implemented using a traditional pattern-based approach. They aimed to improve the modularity, internal code quality, and the flexibility of design patterns. As a result, modularity of the system is improved using design patterns in both cases. However, AOP implementation of design patterns significantly improved the modularity of the system with respect to traditional object-oriented version.

All of those related works investigate the impact of design patterns on maintainability. Most of the results prove that design patterns are able to improve the maintainability of the software. Although there are the result that shows design patterns did not affects the maintainability and understandability at all. That is why the impact of design patterns on software quality attributes is still controversial based on mapping study which is conducted by Ampatzoglou et al. in 2013 [10]. Also, they only investigate the impact of GoF design patterns. Thus, in this study we investigate Patterns of Enterprise Application Architecture (PoEAA) to find out to what extent the impact is on increasing the maintainability.

IV. The Proposed Approach

The research has five main phases: (1) preparation; (2) measurement of existing system (that is the system which is built without considering the use of design patterns specifically, later it is called as non-pattern or NP 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)

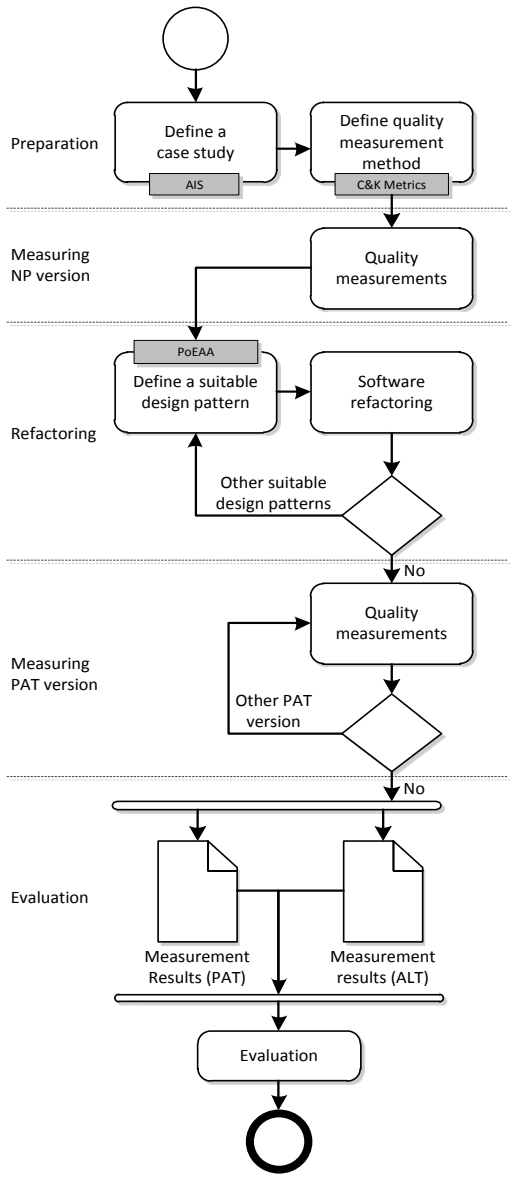

Fig. 2. Activity Diagram of The Proposed Approach.

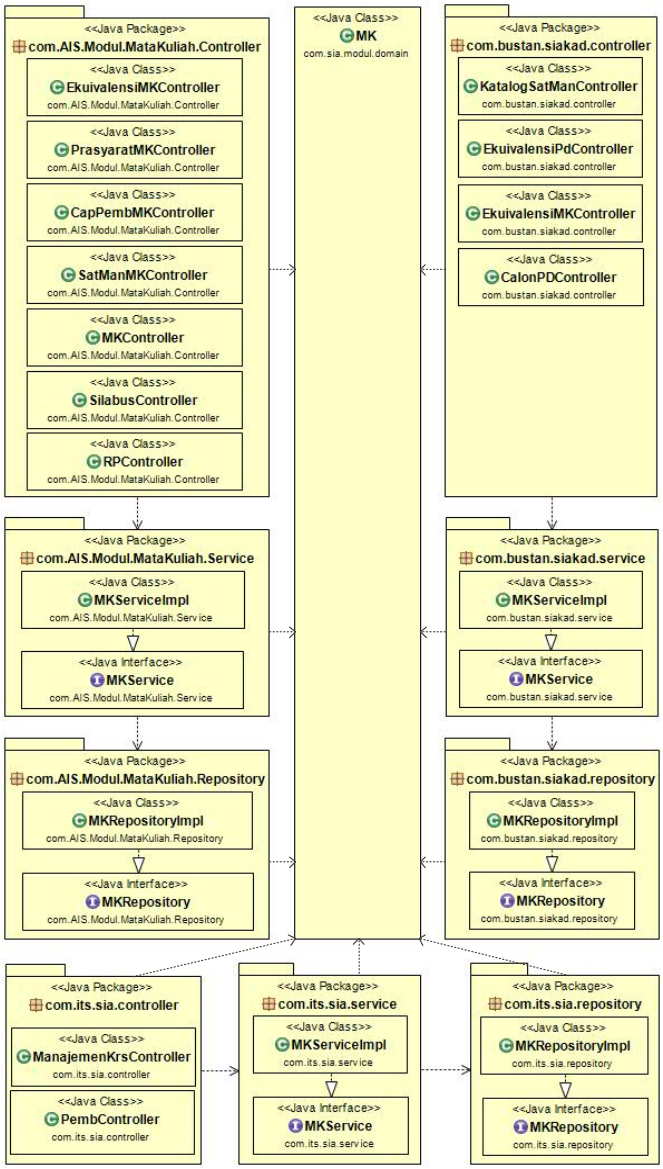

Fig. 3. Class Diagram of The Case Study.

evaluation. We present the research design as an activity diagram in Fig. 2.

A. Preparation

Preparation phase determines several things related to the research requirements and environment, i.e., which part of the AIS to be used as a case study, which design patterns to be utilized, and what quality measurements standards to be applied. The existing version of AIS is categorized as Anemic Domain Model [16] and considered as anti-pattern. It is because Anemic Domain Model has no behavior but setters and getters. It is contrary to object-oriented programming concept where an object is supposed to have both data and behavior. We choose parts of AIS based on the problem which is mentioned in the previous section in order to reduce the duplicated code, coupling, and increase the cohesion. It may increase the maintainability as well. Class diagram of the case study is shown on Fig. 3.

We use C&K metrics [18] and three additional metrics [13] to measure the software maintainability. Those metrics have been used widely [11],[13] and can be used to predict maintainability in general. The metrics have been validated in numerous datasets and techniques. Based on the explanation of each metric, it may represent all of maintainability attributes in ISO/IEC 25010.

B. Measurements

We measure the maintainability of existing AIS prior to refactoring process (called NP version). We use the same methods as NP version to measure PAT version of AIS. Software maintainability is measured by C&K Metrics and three additional metrics of Li & Henry [13]. We use Java metric tool called Chidamber and Kemerer Java Metric [19] and Eclipse Metric Plugin to get metric values. This tool measures the system per class. C&K metrics and additional Li & Henry metrics are described as follows.

1. Weighted Methods per Class (WMC)

WMC is the 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, such that the formula applies as follows. WMC = n if all method complexities are considered to be unity. Where n is the number of methods.

$$WMC = \sum_{i=1}^{n} c_i \quad (1)$$

2. Depth of Inheritance Tree (DIT)

DIT is the inheritance level number of the class, 0 for the root class. This results in consequences as follows. 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)

NOC is the number of direct sub-classes that the class has or number of immediate subclasses subordinated to a class in the class hierarchy. The greater the number of children, the greater the reuse, since inheritance is a form of reuse. The 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.

4. Coupling Between Object Classes (CBO)

CBO is the count of the number of other classes to which it is coupled. 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.

5. Response For a Class (RFC)

RFC is the 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.

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

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

6. Lack of Cohesion in Methods (LCOM)

LCOM is the 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 two 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 ($\{e\}$).

This results in consequences as follows. 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.

This study also uses three additional metrics of Li & Henry [13]. We use NOM, SIZE1, and SIZE2 to predict the maintainability of the systems.

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 another 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.

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.

Mapping is conducted based on which are mentioned explicitly in the literature. Changeability and Modification stability are merged into Modifiability. Table 3 presents the mapping results between the metrics and ISO/IEC 25010 Maintainability sub attributes.

TABLE 3 METRICS MAPPING

Metrics		ISO/IEC 25010 Maintainability
C&K Metrics	WMC	Modularity, Reusability, Modifiability
	DIT	Reusability
	NOC	Reusability
	CBO	Modularity, Reusability, Modifiability, Testability
	RFC	Testability, Modifiability
	LCOM	Modifiability
Li & Henry Metrics	NOM	Modifiability
	SIZE1	Modifiability
	SIZE2	Modifiability

We need to calculate the mean of metric values for all classes which are involved. We also calculate the standard deviation to decide whether mean values are acceptable or not. It is because the value of measurement results is usually not always normally distributed.

We use Median Absolute Deviation (MAD) which 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 [20]:

$$MAD = b\ M_i(|x_i - M_j(x_j)|) \quad (3)$$

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 as follows.

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

C. Refactoring

At refactoring phase, we develop PAT versions by using Domain Logic and Data Source Architectural Patterns because there is a problem with domain model in the existing version of AIS. 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. We name it an alternative version (written as ALT version in Fig. 2).

For 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 utilize two patterns which are Active Record and Data Mapper patterns since 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.

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 contains data, behavior, and data access. We name this version as PAT-AR version. Fig. 4 shows the displacement flow of business logic and repository from each module into domain module.

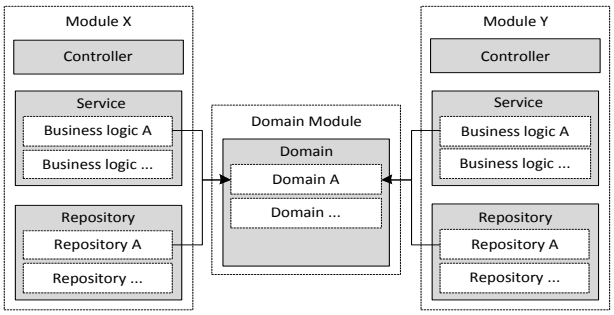

Fig. 4. Displacement Flow of Business Logic and Repository (PAT-AR).

Business logic A from module X and Y are merged into its anemic domain in Domain Module. The same goes with repository A from module X and Y are 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. Fig. 5 shows the architecture of refactored AIS which is PAT-AR version.

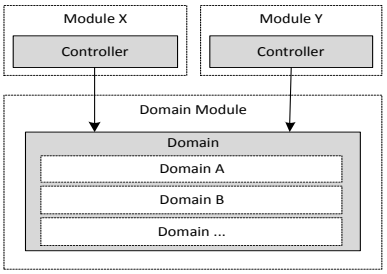

Fig. 5. Architecture of PAT-AR Version.

Domain Model and Data Mapper Pattern

As domain model of existing AIS is anemic, we need to displace business logic from services into domain module. Fig. 6 shows the displacement flow of business logic from service layer into domain model in the domain module.

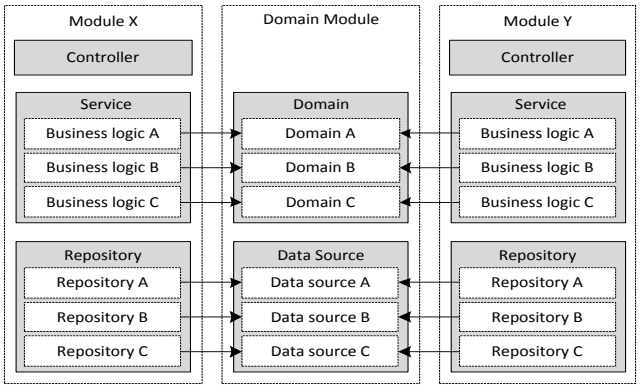

Fig. 6. Displacement Flow of Business Logic and Repository (PAT-DM).

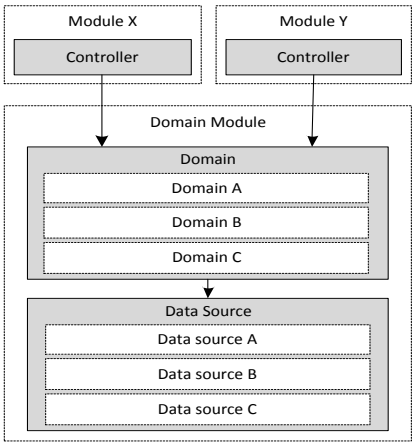

Fig. 7. Architecture of PAT-DM Version.

Business logic of Domain A from service layer in other modules is merged into Domain A in domain module. The same way happens with Domain B, and so on. Thus, it makes the domain model is no longer anemic. 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 may still contain other domain logic and database transaction. If the module uses a unique logic which is only applied on that module, it inherits the related domain model. The same things applies to repository layer. It inherits the related data source from domain module. The modul displacement

process results in a new architecture of refactored AIS as shown in Fig. 7 above.

D. Evaluation

We use Percentage Change or Relative Change proposed by Bennett & Briggs [21] to measure the extent of changes in the evaluation phase.

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

where x represents the starting point of the change, which is NP version in this case, and y represents PAT version of AIS. The relative change is undefined or zero if the value of x equals zero. 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. We analyse the changes based on the explanation of each metric in the literature [18].

V. The Results

This section describes the results of this study. The results are presented based on activity diagram of our proposed approach. There are five phases in this section: (1) preparation; (2) measurements of NP version; (3) refactoring; (4) measurements of PAT versions; and (5) evaluation. The first phase is supposed to be preparation. However, the quality measurements method has been discussed earlier in the previous section. Thus, we only discuss about the case study in the subsection.

A. Preparation

We select some parts of the system as a case study since the whole AIS is too big for the purpose of this study. Case study selection is conducted by selecting one domain model and is then investigated its relationships with other modules. Fig. 3 shows the selected case study which is focused on MK domain model. MK domain model is a courses model of AIS with its attributes, setters and getters.

The basic structure or architecture of the selected case study is the same as explained previously in Fig. 1. Table 4 maps the in-picture module names onto the actual module names.

TABLE 4 MODULE NAME MAPPING

In-picture Module Name	Actual Module Name
com.AIS.Modul.MataKuliah.*	Curriculum
com.bustan.siakad.*	Equivalence
com.its.sia.*	Learning
com.sia.modul.domain	Domain

MK domain model is associated with three modules which are curriculum, equivalence, and learning modules. Each module consists of controller, service, and repository package. There are many 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. Fig. 8 shows the service layer and Fig. 9 shows the repository layer of MK in each module.

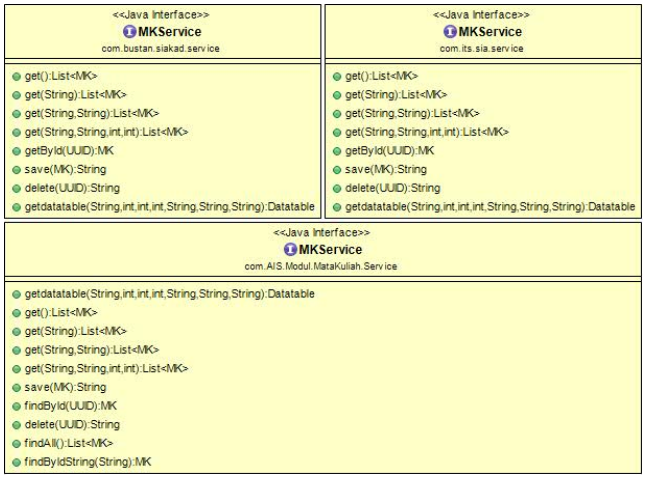

Fig. 8. MK Service Classes.

Based on observation on the module it is found that service layer of MK in equivalence and learning module are 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 are similar. In maintenance process, if there are changes in business process of MK domain, then all of those service and repository classes should be changed.

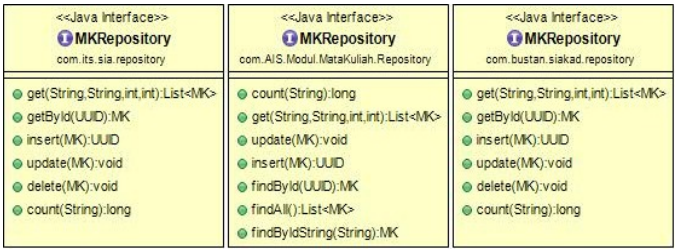

Fig. 9. MK Repository Classes.

B. Measurements of NP Version

Measurement results of NP version of AIS are shown in Table 5. It involves four modules of the architecture as shown in Fig. 3. Classes in three modules: curriculum; equivalence; and learning, consist 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 consists of four classes of presentation layer, two classes of service layer, and two classes of repository layer. Learning module consists of two classes of presentation layer, two classes of service layer, and two classes of repository layer.

Each layer has 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.

C. Refactoring

AIS already uses domain model although still anemic. Thus, we use Domain Model Pattern as its Domain Logic Pattern. Domain model is an object model that incorporates both data and behavior. We move MK business logic which

TABLE 5 MEASUREMENT RESULTS OF NP VERSION

Module	Class	WMC	DIT	NOC	CBO	RFC	LCOM	NOM	SIZE1	SIZE2
Curriculum	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
Equivalence	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
Learning	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
Domain	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**

is located in service layer of curriculum, equivalence, and learning module into the domain model. As shown on Fig. 8, there are eight duplicated methods exist in those modules, thus MK domain model consists of ten methods.

We create two PAT versions using two types of pattern, i.e. Active Record Pattern and Data Mapper Pattern for the Data Source Architectural Patterns because these patterns work well with Domain Model Pattern. They move data between objects and database while keeping them independent. We create a new layer in domain module to hold the database transaction which is data source layer. We create a new class as MK data source in data source layer of domain module. As shown on Fig. 9, there are six duplicated methods exist in those modules, thus MK data source is consist of seven methods. Controller packages in presentation layer, which is part of MVC Pattern, connect directly to the MK class in domain module.

Based on the displacement flow diagrams above, Service and Repository layer from all three modules of AIS, i.e. curriculum, equivalence, and learning module are merged into its domain model in Domain Module. Fig. 10 shows the class diagram of refactored AIS using Active Record Pattern (PAT-AR version), whilst Fig. 11 shows another version of class diagram of refactored AIS using Data Mapper Pattern (PAT-DM version).

Measurements of PAT-AR Version

Table 6 shows the measurement results of PAT-AR version of AIS. This version has 14 classes, which is less than the number of classes in NP 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 relieve the anemic model of the domain. In addition, repository layer is also merged into domain model to conform the Active Record Patterns. Thus, the number of classes in this version is decreased. The class diagram of PAT-AR version is illustrated in Fig. 10 below.

Three modules which are curriculum, equivalence, and learning of this version only consist of presentation layer. Curriculum module has seven controller classes, equivalence module has four controller classes, and learning module has two controller classes.

As mentioned in the previous section, 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 into domain model. Moreover, this version utilizes design patterns where the domain object is no longer anemic.

Measurements of PAT-DM Version

Table 7 shows the measurement results of PAT-DM version of AIS. As seen on Fig. 11, this version has 16 classes which is less than the number of classes in NP 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 conform the Data Mapper Pattern.

Three modules that are curriculum, equivalence, and learning of this version consist only the presentation layer. Curriculum module has seven controller classes, whilst equivalence module has four controller classes, and learning

module has two controller classes. The new layer in domain module, which is data-source layer, is the composite of repository layer from each module.

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.

D. Evaluation

This section presents the extent of changes that occurs between NP and PATs versions of AIS. The following tables show the relative changes of metrics between NP and PATs. These changes involve all classes regardless to its layer. We investigate the relative changes of three values, i.e. sum, mean, and maximum value of all metrics. The changes are illustrated in Table 8, Table 9, and Table 10 respectively.

Almost all changes of sum value (ΔSum) is negative, which indicates a decrease in total complexity of PATs version. Complexity in general, which is represented by metrics, is decreased because the number of classes also decreased. Since PATs has a smaller number of classes, this result from sum point of view is very reasonable and may not represent the general impact of design pattern.

From mean point of view (ΔMean), most of the changes are positive. This indicates that the complexity is increased in PATs. However, this result may also not represent the general impact since some of standard deviation values are larger than its mean and the data is not normally distributed.

Relative change of metrics from maximum value (ΔMax) point of view shows more varied results. Maximum value of CBO, RFC, and SIZE1 are decreased, which means the maximum complexity of all classes regardless to its layer is decreased. However, the maximum value of LCOM is increased on PATs. The increased value is almost double of the NP version. Thus, the next step is to breakdown the result based on its layer to conduct a deeper analysis.

Since some standard deviation values of measurement results are bigger than its mean, we split the evaluation based on the layer. NP version of AIS consists of four layers, i.e. domain, presentation, service, and repository layer. Domain and service layer are merged in PATs, thus it has three layers, i.e. domain, presentation, and data-source layer. We split the evaluation based on three layers because domain and service layer of NP version can be compared with domain layer of PATs, which is not vice versa.

Presentation Layer

Table 11 shows the relative change of metrics values on presentation layer. The number of classes between NP and PATs is the same, i.e. 13 classes. There is no standard deviation value of this layer which is bigger than its mean. Thus we assume that sum, mean, and maximum value may represent the impact of design pattern on software maintainability of AIS.

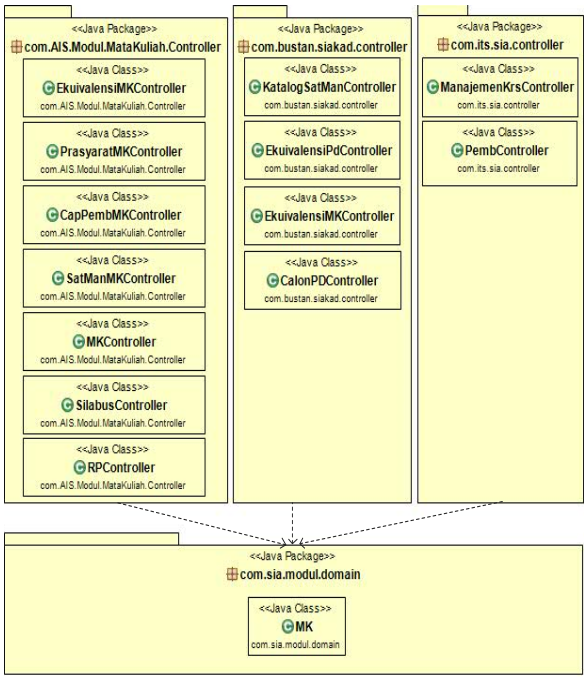

Fig. 10. Class Diagram of PAT-AR Version.

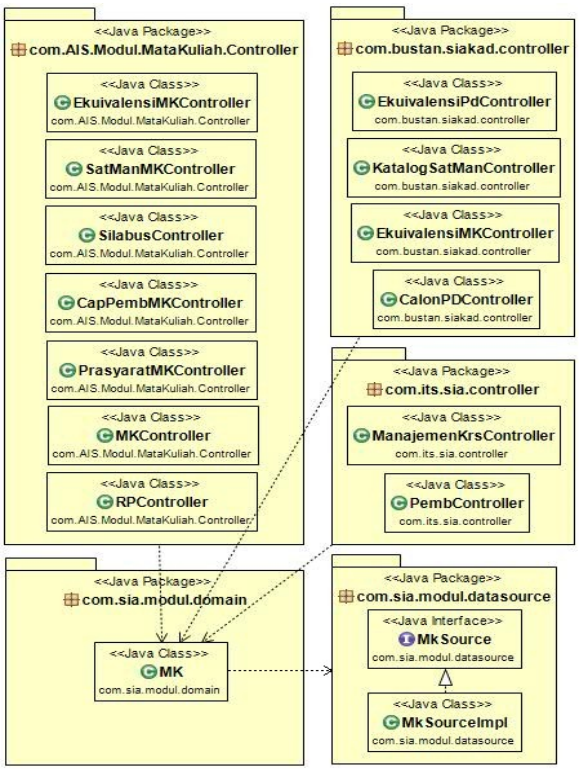

Fig. 11. Class Diagram of PAT-DM Version.

Measurement results of PAT-AR and PAT-DM are the same on this layer. PAT in Table 10 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 NP to PAT versions is decreased. The maximum value of metric is decreased by 1.17% in average. The decreased values occur in CBO, RFC, and SIZE1 metric.

TABLE 6 MEASUREMENT RESULTS OF PAT-AR VERSION

Module	Class	WMC	DIT	NOC	CBO	RFC	LCOM	NOM	SIZE1	SIZE2
Curriculum	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
Equivalence	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
Learning	ManajemenKRSController	18	1	0	45	148	0	16	444	31
	PembController	22	1	0	28	114	113	20	424	31
Domain Data Source	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**

TABLE 7 MEASUREMENT RESULTS OF PAT-DM VERSION

Module	Class	WMC	DIT	NOC	CBO	RFC	LCOM	NOM	SIZE1	SIZE2
Curriculum	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	118	7
	MKController	7	1	0	16	45	7	5	138	10
	RPController	21	1	0	35	132	56	20	507	36
Equivalence	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
Learning	ManajemenKRSController	18	1	0	45	148	0	16	444	31
	PembController	22	1	0	28	114	113	20	424	31
Domain Data source	MK	32	1	0	6	61	442	31	199	45
	MKSource	7	1	0	1	7	21	0	13	0
	MKSourceImpl	8	1	0	6	30	0	7	90	8
Sum		**248**	**16**	**0**	**358**	**1535**	**916**	**219**	**6372**	**354**
Mean		**15.5**	**1**	**0**	**22.38**	**95.94**	**57.25**	**13.69**	**398.3**	**22.13**
Std. Dev.		**8.566**	**0**	**0**	**13.67**	**63.79**	**104.3**	**9.272**	**365.7**	**14.25**
Maximum		**34**	**1**	**0**	**45**	**225**	**442**	**33**	**1364**	**47**

TABLE 8 RELATIVE CHANGES OF SUM VALUES

Metric	Version			ΔSum	
	NP	PAT-AR	PAT-DM	NP→PAT-AR	NP→PAT-DM
WMC	318	234	248	-26.42%	-22.01%
DIT	26	14	16	-46.15%	-38.46%
NOC	0	0	0	0%	0%
CBO	412	354	358	-14.08%	-13.11%
RFC	1750	1509	1535	-13.77%	-12.29%
LCOM	877	927	916	5.70%	4.45%
NOM	248	213	219	-14.11%	-11.69%
SIZE1	6959	6323	6370	-9.14%	-8.46%
SIZE2	399	347	354	-13.03%	-11.28%
				-9.72%	**-8.85%**

The sum of metric values is mostly decreased except for DIT, NOC, and NOM. Total complexity of presentation layer is decreased so that the maintainability is increased. Average value or mean and maximum values of metrics also mostly decreased. The biggest change occurs in CBO. Presentation layer of NP version is related to several service and domain layer. However, presentation layer of PAT version only related to domain layer, which is reduced its coupling between objects. Changes on other metric values are not too significant because there are not many changes occur in classes of presentation layer.

TABLE 9 RELATIVE CHANGES OF MEAN VALUES

Metric	Version			ΔMean	
	NP	PAT-AR	PAT-DM	NP→PAT-AR	NP→PAT-DM
WMC	12.23	16.71	15.50	36.66%	26.73%
DIT	1	1	1	0%	0%
NOC	0	0	0	0%	0%
CBO	15.85	25.29	22.38	59.57%	41.20%
RFC	67.31	107.79	95.94	60.14%	42.54%
LCOM	33.73	66.21	57.25	96.30%	69.73%
NOM	9.54	15.21	13.69	59.50%	43.50%
SIZE1	267.65	451.64	398.13	68.74%	48.75%
SIZE2	15.35	24.79	22.13	61.51%	44.17%
				67.67%	**48.11%**

Modularity of both PAT versions is increased. It is indicated by the decreased value of WMC and CBO metric. Both 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 NP version. The maximum value of CBO is

decreased by 8.16% because in NP version, presentation layer is connected with a domain model and several services. However, in PAT versions, presentation layer only connects with only domain model

TABLE 10 RELATIVE CHANGES OF MAXIMUM VALUES

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

TABLE 4 RELATIVE CHANGES OF METRICS ON PRESENTATION LAYER

Metric	Sum			Mean			Maximum		
	NP	PAT	ΔSum	NP	PAT	ΔMean	NP	PAT	ΔMax
WMC	202	201	-0.5%	15.54	15.46	-0.5%	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	-3.9%	27.62	26.54	-3.9%	49	45	-8.16%
RFC	1439	1437	-0.14%	110.69	110.54	-0.14%	226	225	-0.44%
LCOM	455	451	-0.88%	35.00	34.69	-0.88%	113	113	0%
NOM	181	181	0%	13.92	13.92	0%	33	33	0%
SIZE1	6210	6070	-2.25%	477.69	466.92	-2.25%	1381	1364	-1.23%
SIZE2	308	301	-2.27%	23.69	23.15	-2.27%	47	47	0%
			-1.83%			**-1.83%**			**-1.17%**

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 no more than 4%. Moreover, most of the maximum value does not change. In average, the decreased complexity is only by 1.83% from NP to any pattern versions. The situation is graphically depicted in Fig. 12 below.

Data-source Layer

The comparison of this layer involves only two versions which are NP 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.

The number of classes between NP and PAT version of this layer is not the same. NP version consists of six classes while PAT version consists of only two classes. Number of classes is decreased because of the duplicated code in the existing version, which is mentioned earlier. Standard deviation values of several metrics are also bigger than its mean. Thus, relative change of mean value may not represent the impact of design patterns.

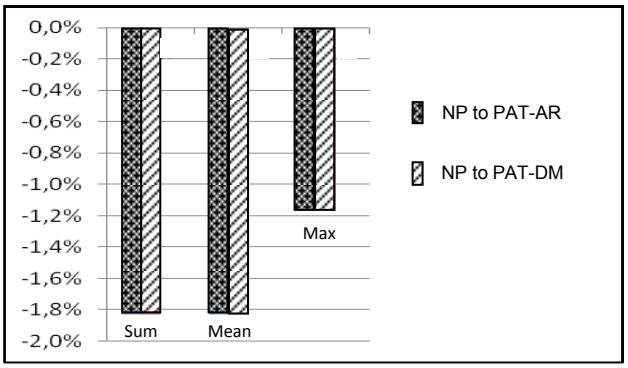

Fig. 12. Relative Change of Metrics on Presentation Layer.

Table 12 shows the relative change of metrics values on data-source layer. We consider using sum value because of the duplicated code. Six classes of NP version basically consist of two distinct classes, i.e. interface and its implementation. Other classes are the duplication of those two classes. If we assume that 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 a hundred of classes with metric values are 10 and 12 respectively. So, the comparison of mean value between App A and App B is 10:11, which means App A is better than App B even though App A has a bunch of duplicated classes, which are more difficult to maintain. If there is a change in App A, then all of hundred classes need to be changed also. 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 this layer.

Most of the sum values are decreased more than a half which means the complexity is decreased. However, there is an increased maximum value of RFC metric. It is because of the service layer of NP version which previously is used to communicate with several duplicated repository classes, yet later is focused on only one class of PAT version. As a result, methods in a single data-source layer class of PAT version are received more calls from domain layer.

From sum point of view, the total number of metric is decreased by 64.22% in average. Thus, we conclude that the

complexity of PAT version on data-source layer is less than NP version. It means the maintainability of NP version on this layer is less than PAT 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 version compared to NP version.

TABLE 5 RELATIVE CHANGE OF METRICS ON DATA-SOURCE LAYER

Metric	Sum			Mean			Maximum		
	NP	PAT	ΔSum	NP	PAT	ΔMean	NP	PAT	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%**

There is one maximum value that is increased, i.e. RFC. However, sum and maximum value itself is decreased in average. PAT is less complex than NP in data-source layer. Thus, PAT has a higher maintainability compared to NP version.

Modularity of PAT 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 PAT 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 NP and PAT version. Thus, zero percent change does not affect the reusability, unless if the change is positive.

Modifiability of PAT 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 class in PAT version contains more methods than NP version. However, the total number of classes in PAT version is less than NP version. That explains why the sum value is decreased.

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

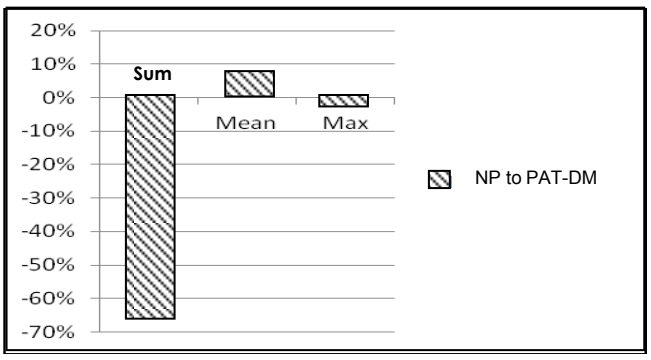

Figure 13. Relative Change of Metrics on Data Source Layer.

In NP version, this layer consists of 19 methods and 12 of them are duplicates. Thus, 63.15% of the method in this version is duplicates. PAT version of this layer is also able to reduce the duplicated methods to a great extent as in domain layer. PAT 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. The situation is graphically depicted in Fig. 13.

Domain Layer

This layer is the comparison between domain and service layer of NP version and domain layer of PATs version. Domain and service layer of NP version consist of six service classes and one domain class. Four of those six service classes are duplicated service classes. The other two are interface class and its implementation. Domain layer of PATs version consists of one class only. There is different number of classes, so we cannot compare them by its mean. We use the relative change of sum and maximum value as in the data-source layer evaluation. We use sum and maximum value because the class in PAT version is basically a composite of classes from NP version.

Table 13 shows the relative change of metrics from NP 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 splitted 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 NP version. However, there is only one class that needs to be handled in PAT-AR version. Meanwhile there are seven classes in NP version. That explains why the sum value is decreased.

TABLE 6 RELATIVE CHANGE OF METRICS ON DOMAIN LAYER (PAT-AR)

Metric	Sum			Mean			Maximum		
	NP	PAT-AR	ΔSum	NP	PAT-AR	ΔMean	NP	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 14 shows the relative change of metrics from NP 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 NP version. However, PAT-DM version also consist of one class only.

TABLE 7 RELATIVE CHANGE OF METRICS ON DOMAIN LAYER (PAT-DM)

Metric	Sum			Mean			Maximum		
	NP	PAT-DM	ΔSum	NP	PAT-DM	ΔMean	NP	PAT-DM	ΔMax
WMC	78	32	-58.97%	11.14	32.00	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%**

Based on both tables, it is concluded that from sum point of view, most of the metric values are decreased. There is one increased metric value namely LCOM. This metric is increased significantly because the domain logic from service layer of NP version is moved into one class along with a bunch of setter and getter. Thus, the communication from classes in other layers is focused on this domain. Most of the maximum value are also increased which mean that the maximum complexity of the class is increased. Maximum value of CBO is decreased because coupling from those duplicated classes is now focused only on domain model in domain layer of PATs version.

As discussed earlier, the amount of effort needed to maintain the most complex classes in NP version is less than any of pattern version. However, the amount of effort to maintain the whole classes of NP version is more than any of PAT versions. The results indicates that PAT-DM version is better than PAT-AR version.

Modularity of both PATs 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 NP version.

Reusability of both PATs 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 splitted 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, future experiment is needed to make the impact more clearly.

Testability of both PATs 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 NP version.

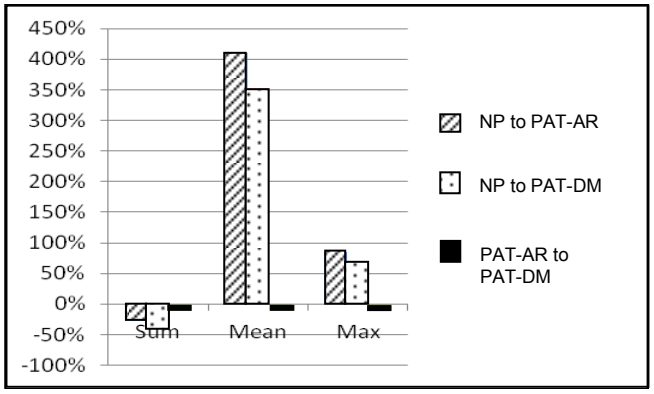

Figure 14. Relative Change of Metrics on Domain Layer.

In NP 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 the case study, PATs 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, PATs 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. The situation is graphically depicted in Fig. 14 above.

VI. ANALYSIS AND DISCUSSION

We analyze the relative change of each metric based on viewpoints of the metrics. The discussion is divided based on its layer.

A. WMC

Table 15 shows the relative change of WMC metric. On presentation and data-source layer, most of the relative change of this metric is negative which mean that the complexity is decreased. More effort is required if this value gets bigger and vice versa. It shows that the effort to maintain the PATs version is less than NP version. Also, PATs version is less application specific than NP version,

thus increasing the possibility of reuse.

On domain layer, sum of the metric is decreased. However, maximum value of the metric is increased. In general, there are fewer classes to maintain in PATs version and it is free from duplicated code. However, fewer classes in PATs version have a higher complexity.

The results conclude that the number and complexity of methods that involved is become a predictor. It predicts the time and effort and 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 limit the possibility of reuse.

TABLE 8 RELATIVE CHANGE OF WMC METRIC

Layer	Sum	Mean	Max
Overall	-22.01%	26.73%	0%
Presentation	-0.5%	-0.5%	0%
Data-source	-60.53%	18.42%	0%
Domain	-58.97%	187.18%	39.13%

B. DIT

Table 16 shows the relative change of DIT metrics. There is no change that occurs in presentation layer because the case study does not contain children classes, thus the depth of its inheritance tree is the same. The complexity of the class' behavior and the potential of the inherited methods to be reused are not change.

The same goes for data-source and domain layer. We cannot add up the DIT metric of all classes. So, there are no changes that occur in all three layers.

TABLE 9 RELATIVE CHANGE OF DIT METRIC

Layer	Sum	Mean	Max
Overall	-38.46%	0%	0%
Presentation	0%	0%	0%
Data-source	-66.67%	0%	0%
Domain	-85.71%	0%	0%

C. NOC

Table 17 shows the relative change of NOC metric. As mentioned earlier, there is no children class in the case study. Thus, there is no change to this metric value. Thus, the reuse and testing requirement of the method of all layers does not change.

TABLE 10 RELATIVE CHANGE OF NOC METRIC

Layer	Sum	Mean	Max
Overall	0%	0%	0%
Presentation	0%	0%	0%
Data-source	0%	0%	0%
Domain	0%	0%	0%

D. CBO

Table 18 shows the relative change of CBO metric. Most of the values are decreased except the maximum value of data source layer. It means that the PATs version is more modular than NP version. There are more independent classes in PATs version. It makes the classes of PATs version is likely easier to reuse it in another application. The maintenance of PATs version is easier to do because the sensitivity to change in other parts of the design is higher.

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.

TABLE 11 RELATIVE CHANGE OF CBO METRIC

Layer	Sum	Mean	Max
Overall	-13.11%	41.20%	-8.16%
Presentation	-3.9%	-3.9%	-8.16%
Data-source	-66.67%	0%	0%
Domain	-81.25%	31.25%	-40%

E. RFC

Table 19 shows the relative change of RFC metric. On presentation layer, all of the relative changes are negative. It means that the complexity of the classes in PATs version is lower than NP version. The testing and debugging of PATs version are less complicated. It is because the tester does not need much effort to understand the code.

On data source and domain layer, the relative change of sum values are decreased which means the complexity of all classes is decreased. However, the maximum complexity of all classes is increased. Although testing and debugging involves fewer classes, some of them are more complicated than NP version.

The results conclude that 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.

TABLE 12 RELATIVE CHANGE OF RFC METRIC

Layer	Sum	Mean	Max
Overall	-12.29%	42.54%	-0.44%
Presentation	-0.14%	-0.14%	-0.44%
Data-source	-65.09%	4.72%	3.45%
Domain	-70.24%	108.29%	7.02%

F. LCOM

Table 20 shows the relative change of LCOM metric. Most of the relative change in presentation and data-source layer is decreased while the maximum values are not change. It means the cohesiveness on PAT version is higher than NP version. Lack of cohesion increases the complexity of the software. It increases the probability of errors during the development phase. Thus, the complexity of PATs version is lower than NP version.

On domain layer, the relative change of LCOM is increased. It seems that PATs version is lack of cohesiveness and the complexity is increased. The increased value of

LCOM on domain layer of PATs version indicates the classes should probably be split into two or more subclasses. However, based on object-oriented programming principal, domain model should contain data and its behavior. So, it is very reasonable that the complexity from setter and getter class to the class which has data and behavior is increased. Moreover, there are no standard of how much metric value is considered to be high or low.

TABLE 13 RELATIVE CHANGE OF LCOM METRIC

Layer	Sum	Mean	Max
Overall	4.45%	69.73%	91.34%
Presentation	-0.88%	-0.88%	0%
Data-source	-58.82%	23.53%	0%
Domain	19.14%	733.96%	91.34%

G. NOM

Table 21 shows the relative change of NOM metric. On presentation layer, there is no change of this metric. The number of method from NP to PATs version remain the same. Thus, there is no change in class complexity.

On data-source layer, the relative change of sum value is decreased. It means PATs version has fewer methods than NP version. The number of methods is decreased because we have eliminated the duplicated classes and methods.

On domain layer, the relative change of sum value is also decreased because of the duplicated methods and classes. Maximum value of the metric is increased. However, we have only one class in domain layer without duplication despite the complexity is increased.

TABLE 14 RELATIVE CHANGE OF NOM METRIC

Layer	Sum	Mean	Max
Overall	-11.69%	43.50%	0%
Presentation	0%	0%	0%
Data-source	-63.16%	10.53%	0%
Domain	-35.42%	352.08%	40.91%

H. SIZE1

Table 22 shows the relative change of SIZE1 metric. Size (LOC) of classes in presentation and data-source layer is decreased. It is caused by the duplicated classes which have been removed in PATs version. The more LOC the class has, then the bigger the effort to maintain the class.

Total line of code in domain layer also decreased because of the duplicated classes. However, maximum LOC of one class is increased because domain model class of PATs version contains both data and behavior instead of a bunch of setter and getter only.

TABLE 15 RELATIVE CHANGE OF SIZE1 METRIC

Layer	Sum	Mean	Max
Overall	-8.44%	48.79%	-1.23%
Presentation	-2.25%	-2.25%	-1.23%
Data-source	-65.2%	4.39%	-5.26%
Domain	-56.07%	207.51%	61.79%

I. SIZE2

Table 23 shows the relative change of SIZE2 metric. The change in sum value of presentation and data-source layer are decreased and there are no changes in its maximum value. It means the total number of methods and number of attributes in PATs version is decreased, thus the complexity is also decreased.

TABLE 16 RELATIVE CHANGE OF SIZE2 METRIC

Layer	Sum	Mean	Max
Overall	-11.28%	44.17%	0%
Presentation	-2.27%	-2.27%	0%
Data-source	-63.64%	9.09%	0%
Domain	-34.78%	356.52%	36.36%

On domain layer, total number of methods and number of attributes is decreased. However, maximum value of the metric is increased because of the domain model in PATs version consist of both data and behavior.

VII. 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 usage 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.

VIII. CONCLUSION AND FUTURE WORK

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. We use nine software metrics to measure the complexity and to predict the software maintainability. There are two design patterns that are used in this study. We use Domain Model as its Domain Logic Pattern. We apply Active Record as well as Data Mapper as its Data Source Architectural Pattern. In the evaluation phase, we calculate the relative change of each metric and evaluate it based on its layer and viewpoints of the metrics.

We compare the measurement results of NP and PATs version based on three layers, i.e. presentation, domain, and data-source. The result is that PoEAA could fix the anemic domain model of AIS. In general, the complexity is decreased as an indicator that the maintainability is increased especially in presentation layer. PoEAA also eliminates the duplicated methods in service and repository layer of NP version of AIS. However, there are several drawbacks as follows.

1) While duplicated classes are eliminated, the maximum complexity value of related layer is increased. The increased complexity occurs in several classes of PATs version.
2) There is lack of cohesion in domain layer of PATs 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.

In future work, we need to strengthen this evidence by conducting an experiment which involve volunteers to maintain AIS. Thus, we can investigate the correlation between the value of software maintainability metrics and the software maintenance activities.

ACKNOWLEDGMENT

Thanks to the parties who have assisted the implementation of this research.

REFERENCES

[1] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*. Pearson Education, 1994.

[2] J. Gonzalez-Sanchez, M. E. Chavez-Echeagaray, R. Atkinson, and W. Burleson, "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*, 2012.

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

[4] A. Christopoulou, E. A. Giakoumakis, V. E. Zafeiris, and V. Soukara, "Automated refactoring to the Strategy design pattern," *Inf. Softw. Technol.*, vol. 54, no. 11, pp. 1202–1214, 2012.

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

[6] M. Fowler, D. Rice, M. Foemmel, E. Hieatt, R. Mee, and R. Stafford, *Patterns of Enterprise Application Architecture*. Addison Wesley, 2002.

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

[8] S. Rochimah, H. I. Rahmani, and U. L. Yuhana, "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*, 2015, pp. 363–368.

[9] F. Handani and S. Rochimah, "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*, 2015, pp. 1–5.

[10] A. Ampatzoglou, S. Charalampidou, and I. Stamelos, "Research state of the art on GoF design patterns: A mapping study," *J. Syst. Softw.*, vol. 86, no. 7, pp. 1945–1964, 2013.

[11] A. Ampatzoglou, G. Frantzeskou, and I. Stamelos, "A methodology to assess the impact of design patterns on software quality," *Inf. Softw. Technol.*, vol. 54, no. 4, pp. 331–346, 2012.

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

[13] W. Li and S. Henry, "Object-Oriented Metrics that Predict Maintainability," *J. Syst. Softw.*, vol. 23, no. 2, pp. 111–122, 1993.

[14] U. L. Yuhana, I. Saptarini, and S. Rochimah, "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 Proceedings*, 2016, pp. 133–137.

[15] Sugiyanto, S. Rochimah, and Sarwosri, "The improvement of software quality model for academic websites based on multi-perspective approach," *J. Theor. Appl. Inf. Technol.*, vol. 86, no. 3, pp. 464–471, 2016.

[16] M. Fowler, "Anemic Domain Model," 2003. [Online]. Available: https://martinfowler.com/bliki/AnemicDomainModel.html. [Accessed: 06-Jul-2017].

[17] A. Ampatzoglou, A. Kritikos, G. Kakarontzas, and I. Stamelos, "An empirical investigation on the reusability of design patterns and software packages," *J. Syst. Softw.*, vol. 84, no. 12, pp. 2265–2283, 2011.

[18] S. R. Chidamber and C. F. Kemerer, "A Metrics Suite for Object Oriented Design," *IEEE Trans. Softw. Eng.*, vol. 20, no. 6, pp. 476–493, 1994.

[19] D. Spinellis, "Tool writing: A forgotten art?," *IEEE Softw.*, vol. 22, no. 4, pp. 9–11, 2005.

[20] Leys, C. et al., "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, 2013.

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

Siti Rochimah is currently a senior lecturer in Department of Informatics Institut Teknologi Sepuluh Nopember, Surabaya Indonesia, and serves as Head of Software Engineering Laboratory. Her areas of expertise are software evolution, software quality, and software engineering in general. She completed his doctorate degree from Universiti Teknologi Malaysia in 2010. She became a Member of IAENG in March 2013.

I Made B. Gautama is currently a junior lecturer in Department of Informatics, STIKOM BALI Indonesia. He was one of the best alumnus of Graduate Programme, Department of Informatics Institut Teknologi Sepuluh Nopember, Surabaya Indonesia. His areas of expertise are software evolution, software quality, and software engineering in general. He completed his master degree in 2018.

Rizky J. Akbar is currently a lecturer in Department of Informatics Institut Teknologi Sepuluh Nopember, Surabaya Indonesia. His areas of expertise are programming, design pattern, and software architecture. He completed his master degree from Ritsumeikan University Japan in 2014. He became a Member of IAENG in 2014.