Measuring Performance Efficiency of Application applying Design Patterns and Refactoring Method

Kholed Langsari[1], Siti Rochimah[1], Rizky Januar Akbar[1]

***Abstract*—Design patterns are always useful concept using in designing and developing a software application. Performance play essential role in the quality attribute of an enterprise application. It is useful to measure and examine how design patterns influence and affect the performance of an application. In this study, we investigate the impact of selected design pattern through refactoring processes for performance efficiency. The systematic study phases included; analyzing, refactoring and performance measuring with implemented in case study SIA system. The performance measuring measure with different test cases and round for the results comparison of each differences test cases and round for design pattern indicate an influence on the performance of an application.**

***Keywords*—Design Patterns, Performance Efficiency, Application Performance, Refactoring, Academic Information System.**

I. Introduction

Software engineering is an engineering discipline for professional and systematic software development rather than individual programming that is concerned with all aspects of software production [1]. It includes aspects such as specification, development, validation, and evolution. The development is concerned of the designing and implementing the software.

Performance is one of the important and essential a quality attribute of software quality [2]. Performance is a non-functional requirement that important factor to consider in enterprise system in order to achieve a high quality of application. A design pattern is commonly thought of as a set of reusable solution to a commonly occurring design problem in object-oriented software as defined by the Gang of Four (GOF) [3]. When applied Patterns, it always comes with some extra layer of indirection and produce characteristic of increased abstraction in the program. This may deliver a positive or a negative impact on performance. Design pattern provides discipline in creating or refactor to a better software structure but they cannot offer any guarantees of the performance of the software quality [4].

There are two main problems addressed by this study: (1) the explosion of impacts in refactoring process using design patterns and (2) the measurement impact of performance efficiency when implemented refactoring technique using design patterns. The purpose of this study is to investigate the impact and influence of design patterns on application quality, especially performance efficiency.

II. Literature Review

A. Software Design

Design principles provide guidance to designers in creating effective and high-quality software solutions. The design is defined as both processes of defining the architecture, component, interface, and other characteristics of a system or component and the result of that process [1]. In the standard list of software life cycle process as ISO/IEC/IEEE Std. 12207-2008 [5], define software design consist of two activities, that are software architecture design and software detailed design.

The Object-Oriented (OO) approach to software design attempts to manage the complexity inherent in real world problems by decomposing the problem into objects and encapsulating it within objects [6], [7]. OO development is a way of program implementation and organized in cooperative collections of objects. Each of object represents a class instance. All classes are hierarchy members of classes connected each other via inheritance relationships.

The most commonly use of model notation for OO approach is Unified Modeling Language (UML). UML is the standard modeling language for object-oriented systems. The UML is a language for modelling and documenting the software-intensive system [8]. Design patterns are often described with UML in various pattern books [9]–[11]. Our case study SIA application implemented object-oriented with Java Enterprise Edition (Java EE) as it main approach and tool in development. SIA has been designing, modeling, and documenting in the standard of UML. In this work, we use UML as our notation and description standard in several phases of methodology and implementation.

B. Design Patterns

Design patterns are defined by Gamma et. al. [11] as simple and elegant solutions to a recurring specific problem arising when designing object-oriented software design. A design pattern provides a guide for refining the subsystems

[1]Kholed Langsari, Siti Rochimah, Rizky Januar Akbar are with Department of Informatics Engineering, Faculty of Information Technology, Institut Teknologi Sepuluh Nopember (ITS), Kampus ITS Sukolilo, Surabaya 60111, Indonesia. E-mail: langsaree@gmail.com; siti@if.its.ac.id; rizky@if.its.ac.id..

or components of a software system and the relationships between them. It describes commonly recurring structures of communicating components that solve a general design problem within a particular context [12].

The Hierarchical-Model-View-Controller (HMVC) is a software architectural pattern. HMVC is a direct extension of the MVC pattern that manages to solve many of the scalability issues [12]. HMVC is a collection of traditional MVC triads operating as one application. Each node is completely self-execution and can process dependently.

Façade is a structural purpose type and object scope design patterns of Gang of Four (GoF). Façade provides the higher-level unified interface (set of the interface) that makes the subsystem easier to use interfaces in a subsystem [11]. Facade simplifies complex code, making it easier to use poorly designed, over-complex subsystems. It is meant to be wrappers around complex functionality, their primary goal is hiding the complexity of an underlying system. Figure 1 described class diagram overview of Façade design pattern.

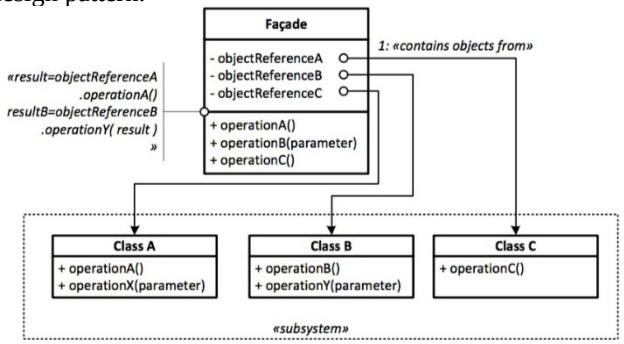

Figure 1. Class diagram of the Façade pattern

The advantages of using Façade are reduced coupling relationships between subsystems, improving maintenance, flexibility and it might increase the performance of the application.

C. Software Quality

Quality is a fundamental property of software systems and generally refers to the degree to which a software system lives up to the expectation of satisfying its requirements [13]. The IEEE Std 1061, the IEEE Standard for a Software Quality Metrics Methodology provides a definition of software quality as software quality is the degree to which software possesses a desired combination of attributes [14]. The ISO/IEC 25010 categorize software quality attributes into eight characteristics. There is functional suitability, reliability, performance efficiency, operability, security, compatibility, maintainability, and transferability.

In this study is focusing on performance efficiency characteristic and sub-characteristic. Performance is concerned with how well the software response when an event occurs [15][16]. The software system events arrive in various patterns which can be characterized as periodic or stochastic. To evaluate whether a system is well performing, the time between the event and the response can firstly be measured, then compared with a previously determined time constraint.

D. Refactoring

In software evolution context, refactoring is a re-engineering technique or the process of changing a software system that aims at reorganizing a program to improve its quality without changing its external behavior [17].

Refactoring (noun): a change that made in the internal structure of software to make it easier to understand and cheaper to modify without changing its external behavior. And Refactor (verb): to restructure software by applying a refactoring processes without changing its external behavior [18]. A refactoring aim to improve a certain quality of system while respect others. Refactoring means improving the design of software without altering its noticeable behavior, developer does not add any new requirement features during the process of refactoring, i.e. they do not do any fixes bug of changes anything about software that would be detected by the software user. Instead, only the internal structure of the technology design of the software is changed [18][19].

Mostly we recognize refactoring and classical refactoring technique for low-level code refactoring that focusing on code level transformation in order to reconstruct of anomalies structures. Knowing how to do refactoring does not mean knowing when to do refactoring. Usually, in identifying when to apply refactoring, we use Design Smells [20] and Code Smells [18] in deciding when system needs to refactoring, and when to stop refactoring.

III. METHODOLOGY

A. Proposed Method

The proposed approach is structured in three fundamental phases, (1) analyzing, (2) refactoring, and (3) performance measuring.

Analyzing phase, (1) we use object-oriented reverse engineering technique [18] focusing on as the process of analyzing a subject system to identify the system's components and their interrelationships and create representations of the system in another form or at a higher level of abstraction. We use several sources of information while reverse engineering, such as read the existing documentation, read the sources code, run the software, use tools to generate the high-level view of the sources code. These sources of information help a lot in analyzing, re-documenting and identifying potential problems of the case study. The result of this activity details of the system such as Architecture View and Class Diagram. (2) Identifying Problem. The result from steps above can be used to determine and identify feasible problems occur in the application. (3) Design Patterns Selection. The analyzing result from previous activities given signs of Code Smell and Design Smell issue related to the legacy software application. The result of this activity is the suitably selected design pattern that going to adapt and implement into the SIA in refactoring process.

In refactoring phase, this activity consists of refactoring and applying design patterns and we follow IMPACT refactoring process model. The process of refactoring is an activity change made to the internal structure of software to make it easier to understand and cheaper to modify without

changing its observable behavior. This activity to apply a design pattern to the legacy system through refactoring technique. The selected design pattern chooses to study and refactor to the system. In this activity, we follow the IMPACT refactoring process model [21].

In performance measuring phase, the dynamic performance efficiency measuring. Dynamic Performance Efficiency Measuring activity uses to measure the performance of both legacies and refactored SIA. The statistics measurement results of legacy and refactored SIA application are carefully compared and differences analyzed.

B. Case Study Application Description

In this study, we are conducted the experiment in particular environment. Our case study is Grading Module of Sistem Informasi Akademik (SIA) [22][23]. The SIA application is an Academic Information System. Academic Information System is an information system with business process for education propose. It consists of various processes and functions handle the education and high education requirement in a systematic way. The SIA application has been manipulating by Faculty of Informatics Engineering, Institut Teknologi Sepuluh Nopember (ITS). The system is design based on modularity, Model-View-Controller (MVC), and Hierarchical-Model- View-Controller (HMVC) architecture. It is implemented Java Enterprise Edition technology using Spring MVC and Hibernate ORM framework. SIA system deploys on Eclipse Virgo and OSGi Framework and PostgreSQL is the main database.

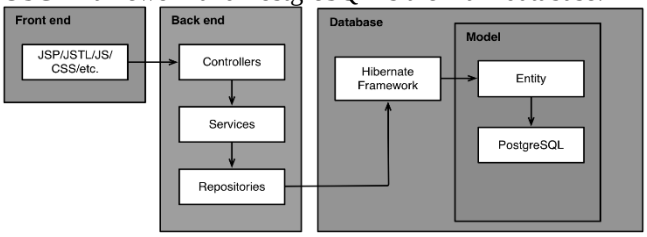

Figure 2. Design of Grading Module Architecture.

The Grading module architecture is to split the project into several logical layers. First, client side, UI layer, in our case study system use HTML/JSP page with JSTL and Spring forms. Second, server side, it is consists of Controller layer (Spring MVC), Service layer (Spring), Repositories (Spring and Hibernate). Third, Data layer: PostgreSQL. Fourth, Model and Java bean classes, which represent application data objects. The design of Grading Module as depicted in figure 2.

Grading Module allows administrator and instructors to submit or change assignment and examination mark, finalcgrades, generate the report, generate student transcript, produce Index and ranking scores, managing questionnaire of courses.

The test scenario uses for performance efficiency measurement is reflect the main functionality user activity of the case study system. In principle, the test activities represent the following data operation: querying, creation, removal, and update, with data querying being the most prevalent activity. The test scenario includes activities typical for this kind of application:

- Listing student details,
- Generating and viewing a report,
- Adding users, and
- Removing users from the application.

E. Performance Efficiency Measure Parameters

There are a number of performance efficiency parameters measured during each test. We have collected parameters that primarily based on ISO standard guideline [16] and support by available tools. The parameters measure includes:

Mean response time, this measurement function used to measure the mean time taken by the system to respond to a user action, where Ai is time between a user's request and a system's response, n is a total number of response events measured. The equation is given in Eq. (1).

$$Main\ Response\ Time = \sum_{i=1 to n} (A_i)/n \qquad (1)$$

Response time conformance, this measurement function used to measure how well does the system response time meet the specified target, where *A* is mean response time result of Eq. (1), and *B* is specified target response time. The equation used to measure is given in Eq. (2).

$$Response\ Time\ Conformance = A/B \qquad (2)$$

Throughput conformance, this measurement function use to measure how well does the throughput meet specified targets, where *A* is a number of tasks completed during the observation time, *B* is observation time period, *C* is target throughput specified, and *n* is a number of observations. The equation used to measure is given in Eq. (3)

$$Throughput\ Conformance = \left(\sum_{i-1\ to\ n} (B_i/A_i)/n \right)/C \qquad (13)$$

Transaction processing error rate, this measurement function used to measure how many concurrent transactions can be processed at any given time against the specified target, where A_i is a number of active transactions at instant given *i*, *B* is total operation duration, and *C* is required transaction processing capacity per unit of time specified. The equation used to measure is given in Eq. (4).

$$Transaction\ Processing = \sum A_i/B/C \qquad (4)$$

Transaction processing error rate conformance, this measurement function used to measure how many concurrent transactions can be processed at a given time against the specified target, where: *A* is the transaction processing error rate result of Eq. 4, and *B* is specified target transaction processed. The equation that use to measure is given in Eq. (5).

$$Transaction\ Processing\ Conformance = A/B \qquad (5)$$

F. Test Environment Specification

The implementation conducted in a controlled environment. The controlled environment gives an accurate result and minimizes noise during run the test for reliable

correct result. The test environment consists of two machines located in a local network, the server and the client side machine. The server and the client side machine located in a local network. The application and database server were located on Ubuntu Server 14.04.3 LTS that run computer server Intel(R) Xeon(R) CPU 2.40GHz 64bits, 7855MB RAM, 500GB HDD with and 1Gbit/s network card. The application server is Java 1.7, with Virgo Server 3.6.4 and PostgreSQL 9.4. The client side was on Macintosh, Intel Mac OS X 10.11. We use Apache's JMeter tool to simulate a number of user activities access to the web application and produce load tasks. We use VisualVM tool for capture data CPU and Memory resource use.

IV. Experimentation and Result

A. Analyzing, Reverse Engineering

The Grading module of SIA application organized Java classes and interfaces by categorized it in packages unique namespace, to represent parts and components of the system. The module consists of three main packages, there are Package Controller, Package Service, and Package Repository followed the basic fundamental design of Spring Web MVC as depicted in figure 3.

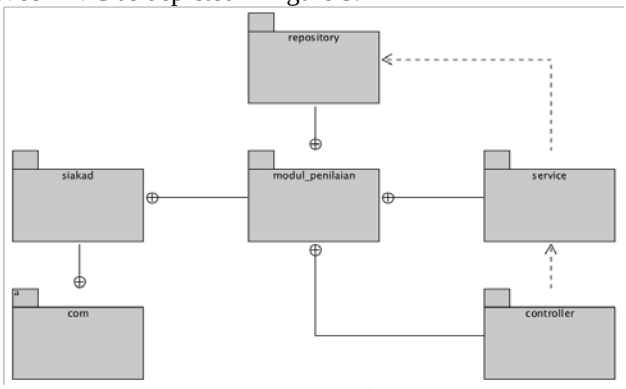

Figure 3. Package Diagram of Grading Module

The interconnection between packages of the module illustrated in package diagram in Fig. 4. The Package Controller (package com.siakad.modul_penilaian. controller), responsible for the act as an interface between Model and View components to process all the business logic and incoming requests, manipulate data using the Model component and interact with the Views to render the final output. The Package Service (package com.siakad.modul_penilaian.service), responsible for the middle layer between presentation and data store. It abstracts business logic and data access. It defines and implements the service interface and the data contracts. The Package Repository (package com.siakad.modul_penilaian.repository), responsible for separating the logic that retrieves the data and maps it to the entity model from the business logic that acts on the model. Mediates between the domain and data mapping layers using a collection-like interface for accessing domain objects. Package repository implemented Repository pattern in interacting with the database through Hibernate Framework as the helper of Data Access Object (DAO).

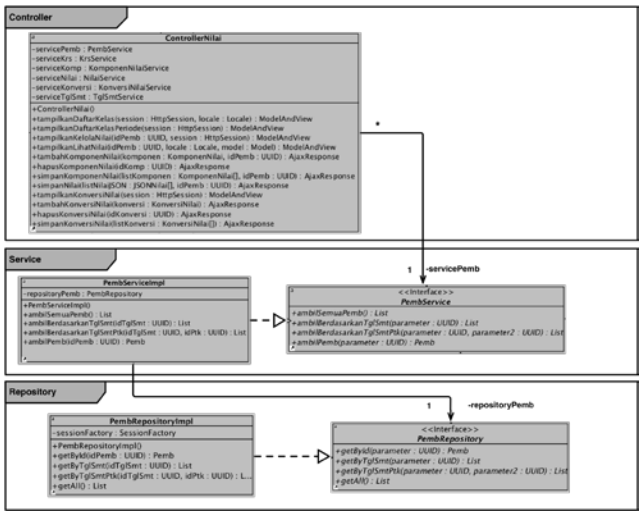

Figure 4. Controller Nilai class diagram interconnection with other classes

B. Identifying Problem

The Grading module is divided into layers follow the layering principle. Layering principle consists of Presentation Layer, Service Layer (the actual business logic) and Data Access Layer. The system source code structure and design is powered and implemented by several technologies and framework as Java EE platform, Spring Framework and Hibernate. And the system is deployed on Virgo server which Apache Tomcat version.

In Grading module, a Controller is typically responsible for preparing a model Map with data and selecting a view name but it can also write directly to the response stream and complete the request. View name resolution is highly configurable through file extension or accepts header content type negotiation, through bean names, a properties ViewResolver file. The model (the M in MVC) is a map interface, which allows for the complete abstraction of the view technology. It can integrate directly with template based rendering technologies as JSP. The model map is simply transformed into an appropriate format in form of JSP request attributes and rendering to user web browser as result of complete request and response.

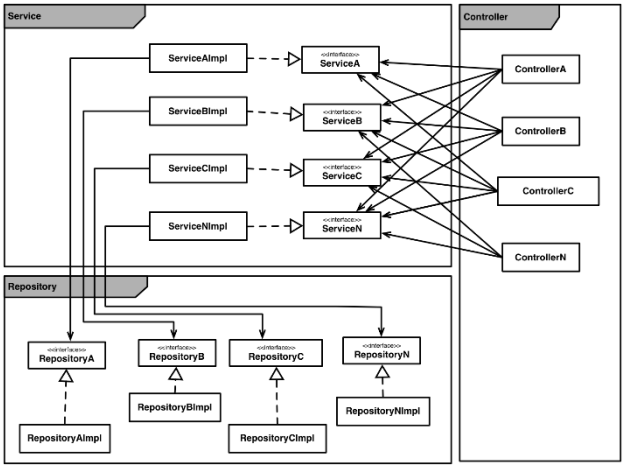

Figure 5. Class diagram of communication and dependencies between classes

In figure 5 classes and subclasses, especially older ones are masses of complex legacy code. When a class in package controller must interact, they often make calls

directly into classes package services. It is creating one-to-many dependencies, and these myriad tendrils of connectivity are difficult to maintain. The subclasses become very delicate since making seemingly insignificant changes in a single subclass can affect the entire program. It creates complexity communication and dependencies between two or more classes or interfaces.

In figure 5 for example, the class ControllerA in package controller, make communication with four interfaces of package service, ServiceA and ServiceB, ServiceC and ServiceN. Class ControllerB and ControllerC and ControllerN also create communication to ServiceA and ServiceB, ServiceC and ServiceN too. It means all Services class have to handle three or more communications at the same time, it is created dependency nightmare for the developer in future maintenance. By this potential emerging problem, we consider to redesign and refactor the module system structure for further feature extent and performance of the system.

C. Refactoring and Applying Design Patterns

According to problem identification section, we chose to introduce the advantages of apply Façade pattern in decreasing dependency and improve the performance of the application. We identify refactoring candidates, plan refactoring activities, implement on planned refactoring tasks, and test to ensure behavior preservation.

We identified the refactoring candidates by introducing to advance Façade design pattern. We created FaçadeService package that contains façade class function that used for functioning easy to use interface communication between classes in package service and package controller.

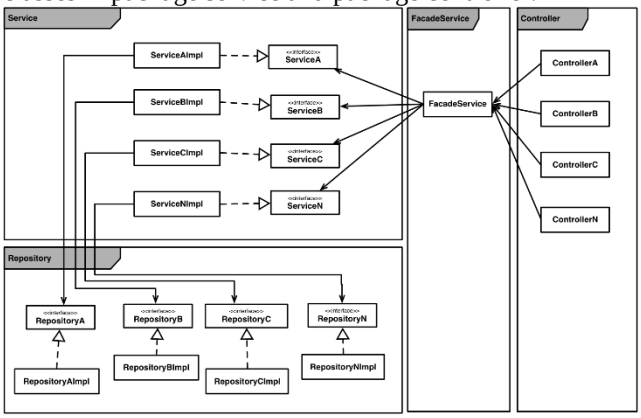

Figure 6. Class diagram of the Façade pattern implement in grading module

In figure 6 present our candidate Façade pattern implemented in grading module. FaçadeService is placed in between the controller and the service. It created opportunities to establish intermediate layers of abstraction with wrap a poorly-designed collection of classes with single well-designed classes that further foster reduced levels of coupling and reduce dependencies of outside code on the inner workings of a library, since most code uses the facade, thus allowing more flexibility in developing the system. This allows the service to remain decoupled from the controller.

This solution is to attain a reduced degree of coupling between services and controller, thereby increasing the freedom and flexibility with which services can be individually evolved. This can result in an elegant architectural design with clean layers of abstraction, but it can also impact impose extra processing overhead that naturally comes with increasing the physical distribution of controller call.

D. Dynamic Performance Efficiency Measurement

There are four test scenarios in the test includes activities typical for the application. There are Test Scenario 1 (TS1): listing student details, Test Scenario 2 (TS2): generating and viewing a report, and Test Scenario (TS3): adding users the application and Test Scenario 1 (TS4): removing users from the application.

All the test scenarios were simulated with the number of concurrent threads (users) increasing from 10 to 350. This final number of threads was determined empirically and it was the maximum number of threads that the application and server could handle (breaking point). After running a full sequence of requests for given number of threads, it was repeated until the total number of requests reached around 3,150 requests. This number was also determined empirically and it was when the response time from the server was stable, meaning that the server had already allocated enough resources to serve a given number of threads.

A test round for one tested case started from simulating 10 concurrent threads. Then the number was set to 10 threads and after that, it was always increased by 10 until the maximum number of 350 threads was reached. Each round was repeated 3 times to ensure that the results are meaningful and reliable.

We defined the Thread Group, Thread Group defines a pool of users that will execute a particular test case against the server. JMeter makes the number of users and the ramp-rate configurable. We use HTTP Request Defaults, HTTP Request Defaults configuration element to the Thread Group. This configuration element sets up the domain IP address of the server, the port and the protocol (HTTP/ HTTPS). We use HTTP Cookie Manager for stores and sends cookies. HTTP Request and the response contains a cookie, the Cookie Manager automatically stores that cookie and will use it for all future requests. For the purposes of this research, the default configurations are enough. We define HTTP Header Manager, it lets us add or override HTTP request headers. The HTTP Cache Manager is used to add caching functionality to HTTP requests within its scope to simulate browser cache feature. Each Virtual User thread has its own Cache. By default, Cache Manager will store up to 5000 items in cache per Virtual User thread. We use HTTP Request element, send an HTTP/HTTPS request to the SIA web server. This configuration element lets us sets up test scenarios as defined, the domain or IP address patch of the web application.

In JMeter performance test application, we set 1,000 threads as target load, 30 minutes Ramp Up Time, 100 Ramp-Up Steps, 10 minutes holding the target rate. This means that the test begins immediately when JMeter starts. In every 0.3 minutes, 10 users will be added until we reach

1000 users. It is can be calculated as 30 minutes divided by 10 steps equals 0.3 minutes per step. 1000 users divided by 100 steps equals 10 users per step. Totaling 10 users every 3 minutes. The first step is 0-10, the second 11-20, and 21-30 etc., because it started 10 threads to run at the beginning. After reaching 1,000 threads all of them will continue running and hit the server together for 10 minutes and all thread will stop.

E. Mean Response Time

The mean response times for the Grading module of SIA application for legacy and refactored are shown in Table 1. The mean response time chart shows how the differences between the implementations. It is clearly visible that Facade had the longest response time throughout the whole measurement. The legacy SIA had an average response time which was about 1.1096 conformance less than for the refactored SIA variant. The differences were visible even after reaching the breaking point of the server.

In table 1 show the different each round and mean result in numeric of the two systems. There are three rounds of the test. The result present in each round of the test by calculating its mean value. For each test case can be summarized in a mean value by calculating all round. The response time measured in millisecond (ms) unit.

TABLE 1.
MEAN RESPONSE TIME

Round No./ metric	Legacy				Refactored			
	TS1	**TS2**	**TS3**	**TS4**	**TS1**	**TS2**	**TS3**	**TS4**
Round1	31685	61700	42681	17932	31885	80710	51252	18642
Round2	29264	64114	41065	14970	28926	67141	45863	19232
Round3	26281	60621	42810	16309	25919	68643	49132	18327
Mean	**26281**	**60621**	**42810**	**16309**	**25919**	**68643**	**49132**	**18327**

F. Response time conformance

The result of response time conformance measure is provided variants. In Table 2 show the result of comparing between legacy and refactored system. The result shows that the refactored system takes much time in response the requests. Response time conformance usually smaller is better and less than 1 is good. The compared test results and test scenarios show that refactored system has a negative impact in response time conformance defectively, except in test scenario 1 (TS1).

TABLE 2.
RESPONSE TIME CONFORMANCE

System/Metric	Test Scenarios				Total
	TS1	**TS2**	**TS3**	**TS4**	
Legacy	26281	60621	42810	16309	26281
Refactored	25919	68643	49132	18327	25919
Conformance	0.9862	1.1323	1.1477	1.1237	0.9862

G. Throughput conformance

Throughput here is calculated as requests/sec unit of time. The time is calculated from the start of the first sample to the end of the last sample. This includes any intervals between samples, as it is supposed to represent the load on the server.

The throughput results also showed clear differences between the investigated design patterns. This is shown in Table 3, result values smaller is better and the default best value is 0. The throughput values remained at an almost constant level until the servers breaking point, and the differences between the implementations also remained proportional. Similar to the results for average response times, the throughput values increased after the simulation passed the breaking point.

TABLE 3.
THROUGHPUT CONFORMANCE

System/Metric	Test Scenarios				Total
	TS1	**TS2**	**TS3**	**TS4**	
Legacy	2.5/sec	2.4/sec	2.5/sec	2.6/sec	10.0/sec
Refactored	2.5/sec	2.3/sec	2.3/sec	2.5/sec	9.6/sec
Conformance	1.000	1.043	1.087	1.040	1.042

H. Transaction processing capacity conformance

The percentage of error reflex the successful requests is depicted in Table 4 and Table 5 for conformance. Based on the results obtained for the percentage of successful and error responses, the measurement point for 350 users was identified as the point where the server could not handle the increased load and the requests resulted in errors. Since the failed requests were not processed entirely, their handling times were shorter compared to the handling times of fully-processed requests. The average response times lowered when the number of simulated users passed the breaking point that the application fully serve.

TABLE 4.
TRANSACTION PROCESSING ERROR RATE

Round No.	Legacy				Refactored			
	TS1	**TS2**	**TS3**	**TS 4**	**TS 1**	**TS2**	**TS3**	**TS 4**
Round 1	9.86 %	41.0 5%	26.9 1%	8.4 5%	7.3 7%	24.4 0%	34.2 7%	6.5 5%
Round 2	10.1 8%	43.0 0%	27.0 2%	7.9 3%	8.2 9%	37.5 3%	16.5 4%	8.5 7%
Round 3	6.86 %	38.8 6%	26.8 6%	6.5 7%	7.2 3%	28.4 8%	22.5 3%	5.6 9%
Mean	8.97 %	40.9 7%	26.9 3%	7.6 5%	7.6 3%	30.1 4%	24.4 5%	6.9 4%

TABLE 5.
TRANSACTION PROCESSING ERROR RATE CONFORMANCE

	Test Scenarios				Total
	TS1	**TS2**	**TS3**	**TS4**	
Legacy	8.97%	40.97%	26.93%	7.65%	21.13%
Refactored	7.63%	30.14%	24.45%	6.94%	17.29%
Conformance	0.85	0.736	0.91	0.91	0.82

V. CONCLUSION

The study aims to measure performance impact on refactoring with design patterns applied on an enterprise software system. The study begins with the importance of performance efficiency of a system application, utilized design patterns in refactoring the legacy system, an enterprise academic information system in Indonesia, go

through the design, analyze and refactoring process, ends at the performance efficiency measuring, and analyze and evaluate the result.

Within the study, the SIA was analyzed and studied through several tools and reverse engineering technique. The Grading module was select to study as the main case of experimentation. The tests carried out 4 test scenarios with respect to the system and implemented the design pattern. All the data gathered shows differences between the compared legacy and refactored application of it implement the design pattern, in terms of performance efficiency. In all the presented tests, the refactored implementation that used Facade pattern had the highest response time and throughput, which resulted in negative impact on performance than the legacy system. The Façade was clearly better in managing design of the system but worse in reduced response time and throughput especially when implemented in the system that already applied another architecture design. The Façade in this case study, it able to reduces transaction processing error rate appreciably.

The presented results are a good starting point for further pattern refactoring implementation. The results can be utilized by application architects and designers to anticipate the behavior of an application depending on chosen design solutions. The results show differences between the legacy and the refactored system using Façade design pattern. However, the conducted tests were limited to only one design patterns and a specific technology, Java EE. Therefore, extended tests should be conducted and cover differences multiple patterns and technologies different from the Java EE technology, for example .NET technology and different several case studies. In addition, the tests should include a wider range of compared design patterns as architecture and other types patterns. The test scenarios also should cover all typical behavior of case study including all use cases of the application. The final objective would be a creation of a set of recommendations containing specific design patterns used on different layers of the application and implemented in various technologies and variants.

REFERENCES

[1] R. S. Pressman, *Software engineering : a practitioner's approach*. New York: McGraw-Hill Inc., 2010.

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

[3] A. Shalloway and J. Trott, *Design patterns explained : a new perspective on object-oriented design*. New York: Addison-Wesley, 2002.

[4] F. Khomh and Y.-G. Guéhéneuc, "An Empirical Study of Design Patterns and Software Quality," in *12th European Conference on Software Maintenance and Reengineering*, 2008, pp. 274–278.

[5] ISO, *ISO/IEC 12207:2008 - Systems and software engineering -- Software life cycle processes*. ISO, 2008.

[6] G. Booch, *Object-oriented analysis and design with applications*, 3rd ed. Redwood City, CA, USA: Addison Wesley Longman Publishing Co., In, 2004.

[7] M. Priestley, *Practical object-oriented design with UML*. New York: McGraw-Hill, 2003.

[8] M. Fowler, *UML distilled : a brief guide to the standard object modeling language*. Boston: Addison-Wesley Professional, 2003.

[9] C. Larman, *Applying UML and patterns : an introduction to object-oriented analysis and design and iterative development*. New Jersey: Prentice Hall PTR, 2005.

[10] M. Fowler, *Analysis patterns : reusable object models*. Boston: Addison-Wesley Professional, 1997.

[11] E. Gamma, R. Helm, R. Johnson, J. Vlissides, and G. Booch, *Design patterns : elements of reusable object-oriented software*. Boston: Addison-Wesley, 1995.

[12] F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, and M. Stal, *Pattern-oriented software architecture : a system of patterns*. Wiley, 1996.

[13] W. Suryn, *Software Quality Engineering : a Practitioner's Approach*. Hoboken, New Jersey: John Wiley & Sons Inc., 2014.

[14] IEEE Computer Society, "IEEE Standard for a Software Quality Metrics Methodology - IEEE Std 1061™-1998 (R2009)," vol. 1998, 2009.

[15] J. Rudzki and T. Systä, "Performance implications of design pattern usage in distributed applications," in *Proceedings of the ISSTA 2006 workshop on Role of software architecture for testing and analysis - ROSATEA '06*, 2006, pp. 1–11.

[16] ISO, *ISO/IEC 25023:2016 - Systems and software engineering -- Systems and software Quality Requirements and Evaluation (SQuaRE) -- Measurement of system and software product quality*. ISO, 2016.

[17] P. Bourque, R. E. (Richard E. . Fairley, and IEEE Computer Society, *Guide to the software engineering body of knowledge*. New Jersey: IEEE Computer Society Press, 2014.

[18] M. Fowler and K. Beck, *Refactoring : improving the design of existing code*. Westford, Massachusetts: Addison-Wesley, 1999.

[19] M. Lippert and S. Roock, *Refactoring in Large Software Projects: Performing Complex Restructurings Successfully*. Wiley & Sons, 2006.

[20] S. G. Ganesh, T. Sharma, and G. Suryanarayana, "Towards a Principle-based Classification of Structural Design Smells," *J. Object Technol.*, vol. 12, no. 2, pp. 1–29, 2011.

[21] G. Suryanarayana, G. Samarthyam, and T. Sharma, *Refactoring for software design smells : managing technical debt*. Morgan Kaufmann, 2014.

[22] S. Rochimah, R. N. E. Anggraini, and H. Rahman, "Rancang Bangun Sistem Informasi Akademik Generik Pada Modul Penilaian Menggunakan Pola Perancangan Hierarchical Model-View-Controlle," Institut Teknologi Sepuluh Nopember, 2015.

[23] U. L. Yuhana, R. J. Akbar, and S. A. Wijaya, "Rancang Bangun Kerangka Kerja Sistem Informasi Akademik Modular Berbasis Web Dengan Pola Arsitektur Hierarchical Model-View-Controller," Institut Teknologi Sepuluh Nopember, 2016.