ISSN 2278-3091
Volume 10, No.3, May - June 2021
International Journal of Advanced Trends in Computer Science and Engineering
Available Online at http://www.warse.org/IJATCSE/static/pdf/file/ijatcse321032021.pdf
https://doi.org/10.30534/ijatcse/2021/321032021

Code Smell Identification As The Basis For Code Refactoring in The Agricultural Information System Portal Case Study at: Gilangharjo Village, Bantul Regency, Indonesia

Argo Wibowo[1], **Lukas Chrisantyo**[2], **Maria Nila Anggia Rini**[3]
[1] Information System, Universitas Kristen Duta Wacana Yogyakarta, Indonesia, argo@staff.ukdw.ac.id
[2] Informatics, Universitas Kristen Duta Wacana Yogyakarta, Indonesia, lukaschris@staff.ukdw.ac.id
[3] Informatics, Universitas Kristen Duta Wacana Yogyakarta, Indonesia, nila@staff.ukdw.ac.id

ABSTRACT

Information system technology continues to evolve over time. Various fields of science including agriculture also utilize information system to improve quality and services in the agricultural sector. Indonesia is an agrarian country where most of the populations work in the agricultural sector. Based on this fact, the Indonesian government is also very supportive towards improving and developing technology in agriculture. Dutatani is one of the Agricultural Information Systems (SIP) that has been consistently developed since 2016 towards precision agriculture. One of the technologies developed is web-based technology which has many sub-systems in it. This raises the problem regarding system scalability wherein each sub-system is developed separately and uses a different development model. Each system uses a specific framework and a native. Therefore this study aims to identify which sub-systems are suitable to be developed and refactored to become a new agricultural information system portal. The identification process used a code smell and metric-based approach. The metrics used were Line of Code (LOC), Complexity, Lack of Cohesion of Methods (LCOM) and God Class. From the results of detection using a code smell, the code detected was 55.17%. This study also revealed a fact that code with a good structure would be easier to detect. Modular code that used a framework and was orderly structured could be read well by detection tools, and showed a high LCOM rate compared to structured and native code.

Key words : Information System, Agriculture, Code Smell, Refactor

1. INTRODUCTION

Information systems are applications with a large scale, which simultaneously developed for a specific purpose. Currently, information systems are required in various sectors, one of which is agriculture. Indonesia is an agrarian country where most of the populations work in the agricultural sector. The Indonesian government is very supportive towards improving and developing technology in agriculture. In addition, the government is also very open to the development of agricultural technology to support the improvement of the quality of agricultural products and an ease in the management of agricultural land. Surely, this can be further supported by the existence of an agricultural information system. The development of information technology in agriculture is highly expected and continuously pursued by the government and the people of Indonesia in order to achieve an increase in agricultural productivity. Such increase can help farmers to reduce agricultural production costs which are considered and deemed ineffective and inefficient.

Dutatani is one of the Agricultural Information Systems (SIP) which has been continuously developed by the development team from the Faculty of Information Technology of Duta Wacana Christian University since 2016 and can be accessed through the main address of http://dutatani.id [1]. This system is called the Integrated Agricultural Information System (IAIS). Such Information System has several modules, namely a precise agricultural land mapping system using satellite map coordinate data on agricultural land [2]. The second module is agricultural activity information system [3], which is a system that records agricultural activities as well as agricultural needs. This system is like a notebook for farmers. The third module is commerce which has a search feature for agricultural product catalogs [4][5]. Users can also sell and buy agricultural products. The fourth module is an agricultural learning information system [6] where farmers and public users can exchange knowledge about agricultural science.

During the applicative use of the website to display various kinds of agricultural information system, several technical problems are found. One problem faced by researchers in the system development is the adaptation of information system integration, there is inefficiency due to the ease of program code adaptation. The information system is not developed in the same framework. This affects the scalability of program code development and poor integration of information system. Another problem is the size of the code that continues to grow as it is developed in a separate framework. This affects the principle of code reusability which refers to a code that can be used together to avoid inconsistencies in input and output data of the information system.

Based on these problems, a system portal that can function as a gateway to the systems that have been previously made is required. The portal is not only intended to combine the existing systems, but also to combine the systems into one main system portal with various modules in it. The information system that was previously stand-alone will be made into a sub-module in the portal. This study aims to find strong evidence as a basis for code refactoring in the agricultural information system portal using code smell identification in the Dutatani program code. After a code smell is found, the next step is to determine the sub-system to be refactored into a new portal.

Code smell is an approach commonly used to identify certain program code in a set of system program code which is bad and needs to be updated [7]. Code smell can detect potentially error code, class with too many objects (God Class), and the level of complexity of the program [8]. The code smell approach applied in this study was metric-based approach.

The study objective is to identify the program code that is not good enough as the basis for code refactoring. This test involved 1 dutatani agricultural portal web application and 4 sub modules in it, namely farmer activity information system, agricultural mapping system, agricultural learning system and agricultural commerce information system. This study also provides a new perspective related to the development of agricultural information system portal by comparing the percentage of code smells on a separate and modular system. The writing structure of the current article is as follows: the first part describes introduction containing the background, the problems found, solutions, study objectives, and contributions of the study. The further section describes the literature review composed based on library sources and references of research articles used in writing this article. Further section is the study methods that discuss the study implementation process starting from metric determination, the detection process, detection results analysis and refactor determination. This end section of this article is the conclusions and limitations of the study which are intended for further development.

2. LITERATURE REVIEW

2.1 Code Smell

Code Smell is a software development problem that occurs at a low level (code) [9] in the form of a collection of “bad” codes in design and implementation [10]. Code smells can be identified from the following things, namely: feature envy, message chain, god class, and long method [7] [9] [11]. The impact of code smells is the degree of adaptation, ease of understanding of program code, scalability, maintainability and low reusability [9] [12].

Studies on Code smell have been conducted several times, including measuring the likelihood of a code smell [7]. The pervious study proposed a formula to measure the likelihood of a code smell with several parameters such as long methods, conditionals, and code clones. Another study explained the visualization of code smells as a software change control [13]. This study is intended to assist developers to easily find out the codes to be changed in more efficient costs and time. Another study observed relationship between representation of different subjects and identification of code smells [14].

2.2 Metric-Based Approach

Metric-based approach is an approach that allows us to measure the quality of a data model [15]. Metrics can help in making decisions and strategies [16]. Code smell analysis can also be applied by using metrics. A metric-based approach for code smells is applied by counting the following factors [9]:

1. Lines of Code (LOC),
2. Number of Attributes per Class (NOA),
3. Number of Methods per Class (NOM),
4. Weighted Methods per Class (WMC),
5. Coupling Factors (CF),
6. Lack of Cohesion of Methods (LCOM),
7. Loose Class Cohesion (LCC),
8. Method Hiding Factor (MHF),
9. Number of Children (NOC),
10. Depth of Inheritance Tree (DIT) .
11. God Class
12. Complexity

To know the metric value in a code smell, a cutoff value is needed which aims to interpret a good and true value [17]. Nearly perfect cutoff value is obtained from 2 main sources, namely statistical and semantic information [18]. Source derived from statistical information depends on the needs to be measured using the Java and C ++ systems with metrics (Michele Lanza and Radu Marinescu, 2006):

1. Average Number of Methods (NOM)
2. Average Lines of Code (LOC)
3. Average Cyclomatic Number (CYCLO)

This study assembled application-based metrics that had been made in previous studies. With a large number of applications, the lines of code and complexity of the application became large, so the researchers used simple techniques to determine each metric, namely: a typical value that covered most of the data, a lower margin that was higher than the interval, and a value that exceeded the value considered as a sample.

This study used 2 statistical methods in determining high and low values, namely the average (AVG) and standard deviation (STDEV). To find out the AVG and STDEV values, it can be assumed that they were normally distributed. In addition, the researchers also recognized the three margins of the typical value for each metric, namely:

1. Lower margin: AVG-STDEV
2. Medium Margin: AVG+STDEV
3. Very high margin: (AVG+STDEV)*1.5

Researchers considered the value as very high if it was 50% higher than the cutoff value for the medium margin.

2.3 Agricultural Information System Portal

Information system (IS) is a set of information technology (IT) applications integrated into a large system which have specific goals [19], and interacts with various users both internal and external users[20][21]. A good information system is able to support the business processes of the organization [22]. Sometimes, information system on a large scale have many features or modules within. Although these features are different, they still have the same purpose as the parent information system. This of course requires an information system portal [23] so as to make it easier for users to use all features of the information system and explore one feature to another without losing navigation.

Information systems with a portal model require good integration so that the functions between systems can communicate well with each other. Good integration can increase scalability [24] so that the system can be developed quickly and easily (so called maintainability). Maintainability is important in integration since complex systems require high maintainability [25].

Previous studies have produced farmer activity information [3], land mapping [2][26], commerce [4][5] and learning [6] systems. These 4 systems will be combined into 1 new portal, since the old portal has not been well integrated. In order to become a well-integrated agricultural information system portal, code refactoring that supports scalability and maintainability is required. Code refactoring is required when the code loses structure which leads to difficulty in reading the design due to a lot of duplications in the code [27][28]. In addition, we can understand the structure of the program better by solving bugs so as to maintain the speed of software development [27] [29].

Code refactoring in this study is required due to the following reasons:

1. Large-scale study developed by a team consisting of several lecturers and students, each of whom departed from different coding habits. Code refactoring is needed to clear up and eliminate duplication of existing functions.
2. Through code refactoring, the structure of codes developed by various team members will be traced in more depth, which indirectly also traces the codes with the potential to cause bugs. Thus, these bugs can be prevented or fixed.
3. Since this system is expected to be large software, ease of code maintenance is therefore important. The results of the first and second factors above are attempted to increase readability and reduce bugs to achieve a good coding design. Good design is essential in accelerating software development.

Before getting into the code refactoring process, a code smell analysis is needed so that it can detect the codes that need to be fixed and need quality improvement [17].

Code smells on integrated and continuous information systems often occur due to the lack of the developer's ability to understand the advanced code created by the previous team [30]. This can be overcome by implementing the same framework [31] on the portal information system and dividing other information systems into several modules (Md Shariful Haque, Jeff Carver, Travis Atkison, 2018. Causes, Impacts, and Detection Approaches of Code Smell: Survey Proceedings of the ACMSE 2018 Conference, No. 25, 1-8.). This study contributes to the importance of code smell detection in information systems with different code structures, and serves as the basis for code refactoring.

3. STUDY METHOD

The study method applied here was divided into 4 stages as shown in Figure 1.

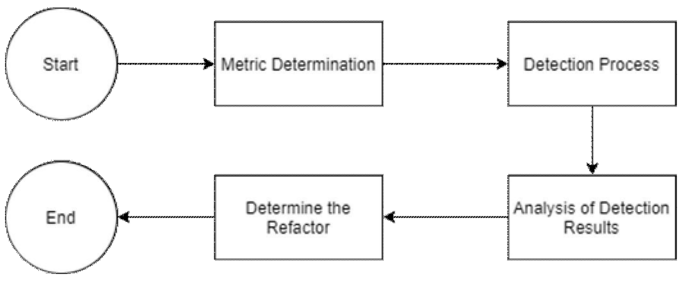


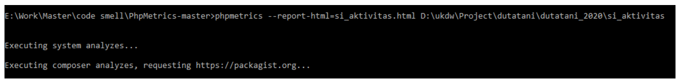
Figure 1: The Study Stages

3.1 Metric Determination

This study used a metric approach [9] to measure the level of need for Code Refactoring. The software used was PhpMetrics because the information system is made with various approaches to the PHP programming language. The metrics used were the Line of Code (LOC) or commonly called the Logical Line of Code (LLOC), Complexity, Lack of Cohesion of Methods (LCOM), God Class because it is 90% more accurate in detecting code smell level [9]. Line of Code is an executable and declarative line of code. A line of code refers to one or more statements followed by line-ending comments. A full line comment is not a line of code. Blank lines (or lines containing only whitespace characters) are also not lines of code. Complexity refers to the level of complexity of the program structure. The value of Cyclomatic complexity must remain below 10. This expresses well-structured and well-written code, high trialability, low cost and effort to build and maintain. If the number of Cyclomatic complexity is above 10, the source code is complex, has low testability, and high costs and efforts to build and maintain [32]. LCOM Metric measures the cohesion or attachment to structural information taken entirely from its source code (for example, attribute references in methods and method calls) which captures the extent to which class elements are shared [33]. High LCOM values mean low cohesion. LCOM values vary from 0 to 4 [34]. God class is a class that has too many functional lines and delegates small details to other classes [35]. Object oriented metrics are used to measure the properties of object-oriented software applications [34].

3.2 Detection Process

It was conducted by running PhpMetrics tools on 4 information systems and 1 old web portal. Information systems were detected one by one and the results were viewed one by one then averaged. Figure 2 shows an example of running a PhpMetrics script on one of the data management modules of the agricultural activity information system. PhpMetrics would then generate a new directory containing code smell analysis reports in the form of a complete html file. This process was run 5 times to track code smells in old agricultural IS portal as well as agricultural activity, land mapping, and learning.



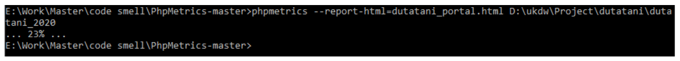
```
E:\Work\Master\code smell\PhpMetrics-master>phpmetrics --report-html=si_aktivitas.html D:\ukdw\Project\dutatani\dutatani_2020\si_aktivitas

Executing system analyzes...

Executing composer analyzes, requesting https://packagist.org...
```

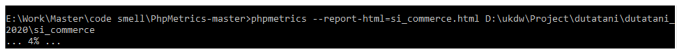
Figure 2: PhpMetrics Detection Process

There were 2 of 5 systems that were failed to be identified during detection processes, namely the old IS Portal and Commerce IS. It can be observed in Figures 3 and figure 4 that PhpMetrics stopped at 23% for the old IS Portal and at 4% for Commerce IS. This indicated that the codes created in it were not formatted properly.



```
E:\Work\Master\code smell\PhpMetrics-master>phpmetrics --report-html=dutatani_portal.html D:\ukdw\Project\dutatani\duta
tani_2020
... 23% ...
E:\Work\Master\code smell\PhpMetrics-master>
```

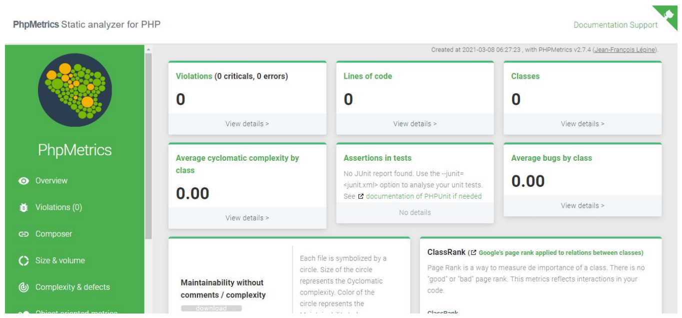
Figure 3: Failure in Detecting Old IS Portal



```
E:\Work\Master\code smell\PhpMetrics-master>phpmetrics --report-html=si_commerce.html D:\ukdw\Project\dutatani\dutatani_
2020\si_commerce
... 4% ...
```

Figure 4: Failure in Detecting Commerce IS

After the detection process on 5 Agricultural IS was complete, the resulting report can be seen in Figures 5, 6 and 7. Only 3 reports were generated because the Old IS Portal and Commerce IS had failed detection.



PhpMetrics Static analyzer for PHP
Documentation Support

Created at 2021-03-08 06:27:23 , with PHPMetrics v2.7.4 (Jean-François Lépine).

PhpMetrics
Overview
Violations (0)
Composer
Size & volume
Complexity & defects

Violations (0 criticals, 0 errors)
0
View details >

Lines of code
0
View details >

Classes
0
View details >

Average cyclomatic complexity by class
0.00
View details >

Assertions in tests
No JUnit report found. Use the --junit=<junit.xml> option to analyse your unit tests. See documentation of PHPUnit if needed
No details

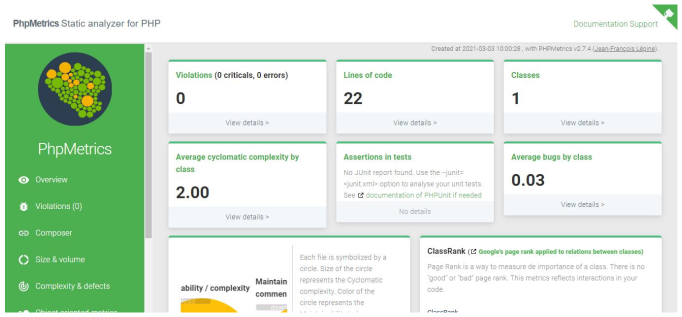
Average bugs by class
0.00
View details >

Maintainability without comments / complexity
download

Each file is symbolized by a circle. Size of the circle represents the Cyclomatic complexity. Color of the circle represents the

ClassRank (Google's page rank applied to relations between classes)
Page Rank is a way to measure de importance of a class. There is no "good" or "bad" page rank. This metrics reflects interactions in your code.

Figure 5: Report on Agricultural Activity IS Detection



PhpMetrics Static analyzer for PHP
Documentation Support

Created at 2021-03-03 10:00:28 , with PHPMetrics v2.7.4 (Jean-François Lépine).

PhpMetrics
Overview
Violations (0)
Composer
Size & volume
Complexity & defects

Violations (0 criticals, 0 errors)
0
View details >

Lines of code
22
View details >

Classes
1
View details >

Average cyclomatic complexity by class
2.00
View details >

Assertions in tests
No JUnit report found. Use the --junit=<junit.xml> option to analyse your unit tests. See documentation of PHPUnit if needed
No details

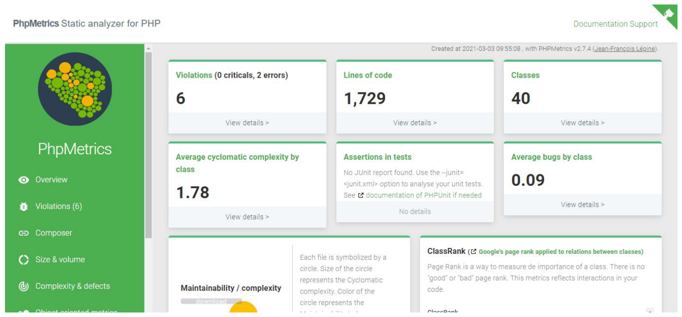
Average bugs by class
0.03
View details >

ability / complexity
Maintain commen

Each file is symbolized by a circle. Size of the circle represents the Cyclomatic complexity. Color of the circle represents the

ClassRank (Google's page rank applied to relations between classes)
Page Rank is a way to measure de importance of a class. There is no "good" or "bad" page rank. This metrics reflects interactions in your code.

Figure 6: Report on Land Mapping IS Detection



PhpMetrics Static analyzer for PHP
Documentation Support

Created at 2021-03-03 09:55:08 , with PHPMetrics v2.7.4 (Jean-François Lépine).

PhpMetrics
Overview
Violations (6)
Composer
Size & volume
Complexity & defects

Violations (0 criticals, 2 errors)
6
View details >

Lines of code
1,729
View details >

Classes
40
View details >

Average cyclomatic complexity by class
1.78
View details >

Assertions in tests
No JUnit report found. Use the --junit=<junit.xml> option to analyse your unit tests. See documentation of PHPUnit if needed
No details

Average bugs by class
0.09
View details >

Maintainability / complexity
download

Each file is symbolized by a circle. Size of the circle represents the Cyclomatic complexity. Color of the circle represents the

ClassRank (Google's page rank applied to relations between classes)
Page Rank is a way to measure de importance of a class. There is no "good" or "bad" page rank. This metrics reflects interactions in your code.

Figure 7: Report on Learning IS Detection

3.3 Analysis of Detection Results

As explained in the previous section, the detected code smells were Line of Code (LOC), Complexity, Lack of Cohesion (LCOM) and God Class. The systems listed in the table were the old SI Portal which accommodates all sub-systems, Agricultural Activity IS, Learning IS, Land Mapping IS and Commerce IS. Learning SI is SI which will be used as a new home to replace the old portal. Old IS Portal uses code igniter framework, Agricultural Activity IS uses native PHP with self-made object class, Learning IS uses Laravel framework, Land Mapping IS uses PHP Native and Commerce IS uses

PHP Native with self-made objects and structures. Table I shows the LOC results for the 5 tested systems and subsystems.

Table 1: LOC Metric

System	Number of LOC
Old Portal	Failed
Old Portal Only	37
Agricultural Activity	0
Learning	25
Land Mapping	20
Commerce	Failed

Tables 1 to 4 show the failed results for Old Portal and Commerce testing. This occurred because the system being tested was so complex that PhpMetrics was unable to resolve code smell searches. The code that was traced did not have a clear structure since it used different native PHP and frameworks. Line of Code in Learning IS could be calculated well with an average value of 25 lines per code that can be executed. Furthermore, Land mapping IS which used PHP Native had an average value of 20 lines per code that can be executed. Agricultural Activity IS used the PHP Native object so that it could not be recognized by PhpMetrics and obtained a score of 0.

Table 2: Complexity Metric

System	Number of Complexity
Old Portal	Failed
Old Portal Only	148.59
Agricultural Activity	0
Learning	46.4
Land Mapping	4.33
Commerce	Failed

Table 2 shows the Complexity of the metrics in the 5 tested systems. The Agricultural Activity IS got a value of 0 again because of the code structure couldn't be read by PhpMetrics. Learning SI obtained a score of 46.4, which indciated that the code structure was quite complicated because it used a framework. Framework file certainly affected test score. On the other hand, Land Mapping IS obtained a score of 4.33, which indicated that the code was well structured and well written. This was due to Land Mapping IS used PHP Native with a very simple structure.

Table 3: LCOM Metric

System	Number of LCOM
Old Portal	Failed
Old Portal Only	2.23
Agricultural Activity	0
Learning	2.63
Land Mapping	1
Commerce	Failed

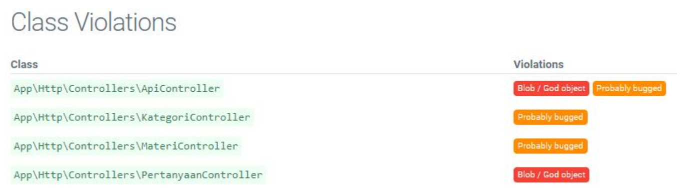
Table 3 shows the Lack of Cohession of Methods of the metrics in the 5 tested systems. Land Mapping IS obtained a score of 2.63, which indicated that the code had a low relationship when compared to Learning IS which only obtained a score of 1. This indicated that Learning IS had a good value due to the low level of relationship with the MVC (Model View Controller) framework pattern. This will be interesting when Learning IS is compared to Old Portal without any sub-systems in it so that they both have a clean structure without any sub-systems. Both Old Portal and Learning IS used different frameworks.

Table 3 shows the Lack of Cohession of Methods of the metrics in the 5 tested systems. Land Mapping IS obtained a score of 2.63, which indicated that the code had a low relationship when compared to Learning IS which only obtained a score of 1. This indicated that Learning IS had a good value due to the low level of relationship with the MVC (Model View Controller) framework pattern. This will be interesting when Learning IS is compared to Old Portal without any sub-systems in it so that they both have a clean structure without any sub-systems. Both Old Portal and Learning IS used different frameworks.

Table 4: GOD Class Metric

System	Number of God Class
Old Portal	Failed
Old Portal Only	198
Agricultural Activity	0
Learning	2
Land Mapping	0
Commerce	Failed

God Class denotes a class code which contains a complex function or logic. Based on table 4, there were 2 God Classes in Learning IS and there was no God Class in Land Mapping IS. Based on the PhpMetrics report as shown in Figure 8, it can be seen that the God Class was the controller class. It is natural to see the MVC architecture used, so that 1 class controller can relate to many class models and views. Therefore PhpMetrics will detect the class controller to be God Class. In contrast to Land Mapping IS which still used PHP native, there were almost no classes that contain many functions in it.



Class Violations

Class	Violations
App\Http\Controllers\ApiController	Blob / God object, Probably bugged
App\Http\Controllers\KategoriController	Probably bugged
App\Http\Controllers\MateriController	Probably bugged
App\Http\Controllers\PertanyaanController	Blob / God object

Figure 8: God Classes in Learning IS

The results of code smell detection presented in tables 1 to 5 indicated that the Old IS Porta, Agricultural Activity and Commerce could not be detected by PhpMetrics. Therefore, the three ISs would be eliminated from the new Portal candidate. Old portal only, Learning and Land Mapping remained here. Each of these three SIs was unique in that the Old Portal and Learning used a framework, and land mapping used a native. The use of frameworks will certainly further help the development process [36] for several reasons, namely 1) it is most often used in modern website development, 2) it is a form of reusing existing functions, 3) frameworks can solve the standardization problem of writing code during the IS development process. Therefore, the two systems that use the framework, namely SI Old Portal and SI Learning woud be further compared. The comparison can be seen in table 5.

Table 5: Comparison Between Old Portal and LEARNING as New Portal Metric

Code Smell	Old Portal	Learning
LOC	37	25
Complexity	148.59	46.4
LCOM	2.23	2.63
God Class	198	2

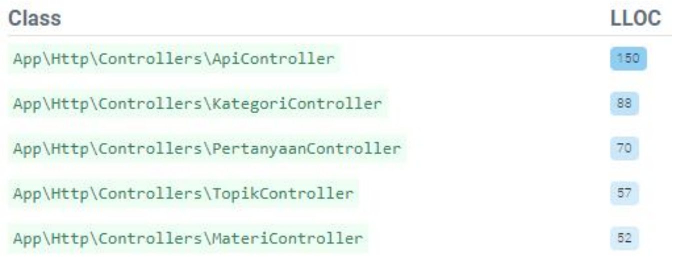
Based on the result of comparison between Old IS Portal and Learning IS, it can be seen that the Learning IS values were better than Old IS Portal. LOC value for Old Portal was 37 and for Learning IS it was only 25. Complexity values between the two was also quite different namely 148.59 for Old IS Portal and 46.4 for Learning IS. Based on LCOM value, it was also shown that the Learning IS obtained a higher value than OLD IS Portal (2.63>2.23). The God Class aspect also showed a significant difference namely 198 and 2. Based on the results above, it can be determined that the IS that would be refactored and made into a new portal was Learning IS since its 4 metric code smells showed better results than Old IS Portal.

3.4 Determine the Refactor Targets

The further stage was to determine the classes to be refactored. There were 4 parts as the basis of the refactoring process, namely the LOC, Complexity, LCOM and God Class metrics.

A. LOC

Figure 9 shows the results of LOC identification using PhpMetrics. In this study, the top 5 classes with the highest number of LOC were taken. This class is a controller class.



Class	LLOC
App\Http\Controllers\ApiController	150
App\Http\Controllers\KategoriController	88
App\Http\Controllers\PertanyaanController	70
App\Http\Controllers\TopikController	57
App\Http\Controllers\MateriController	52

Figure 9: Class with the Highest Number of Line Code in Learning IS

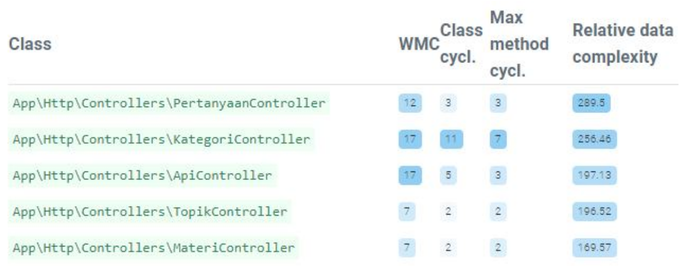
It was different with the LOC in Old Portal which was dominated by the default framework classes. Therefore, it is necessary to perform manual filtering to obtain 5 IS classes as listed in table 6 below:

Table 6: LOC in Old Portal IS

Class	LLOC
Change	221
Forum	213
Discussion	180
Login	177
User	177

B. Complexity

There were 5 classes with the highest complexity level in Learning IS as shown in Figure 10. The detected class was the controller class, the same class as obtained in LOC detection.



Class	WMC	Class cycl.	Max method cycl.	Relative data complexity
App\Http\Controllers\PertanyaanController	12	3	3	289.5
App\Http\Controllers\KategoriController	17	11	7	256.46
App\Http\Controllers\ApiController	17	5	3	197.13
App\Http\Controllers\TopikController	7	2	2	196.52
App\Http\Controllers\MateriController	7	2	2	169.57

Figure 10: Class with the Highest Complexity Level in Learning IS

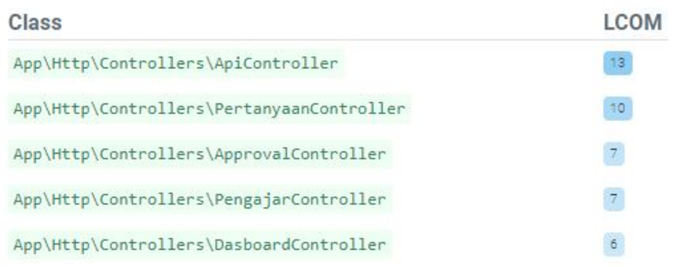
Complexity in Old Portal OS was dominated by the default framework classes. Therefore, it is necessary to perform manual filtering to obtain the classes as listed in table 7 below:

Table 7: Complexity LEVEL IN Old Portal IS

Class	**LLOC**
Forum	961.06
User	900.23
Change	841.16
Discussion	576.71
Admin	400.08

C. LCOM

Figure 11 shows the 5 classes with the highest LCOM levels in Learning IS. A high LCOM level showed a good indicator, but it should be considered that this class also had a high code line level and was included in the God Class category as shown in Figure 8



Class	LCOM
App\Http\Controllers\ApiController	13
App\Http\Controllers\PertanyaanController	10
App\Http\Controllers\ApprovalController	7
App\Http\Controllers\PengajarController	7
App\Http\Controllers\DasboardController	6

Figure 11: Classes with the Highest LCOM level in Learning IS

There was no LCOM in Old Portal since LCOM was detected in the default framework class. For this reason, there was no Old Portal class that should be refactored in the LCOM metric.

D. God Class

List of God classes can be seen in Figure 8. The classes declared large were the ApiController, CategoryController, MaterialController, and QuestionController classes. In Old Portal IS, most of the classes detected were the default framework classes. Therefore, if it was filtered manually, 5 classes were found as listed in table 8 below:

Table 8: God Class SI Old Portal

Class	**Violation**
Admin	Probably Bugged
AdminNews	Probably Bugged
AdminChange	Probably Bugged
Forum	Probably Bugged
Change	Too complex class code Too complex method code Probably bugged

Based on the data obtained, the classes that required refactoring are listed in table 9 below:

Table 9: Refactor Classes

No	Class	Code Smell			
		LCOM	Complexity	LOC	God Class
1	ApiController	v	v	v	v
2	Question Controller	v	v	v	v
3	Category Controller		v	v	v
4	MaterialController		v	v	v
5	TopicController		v	v	
6	Approval Controller	v			
7	Instructor Controller	v			
8	DashboardController	v			
9	Change		v	v	v
10	Forum		v	v	v
11	Discussion		v	v	
12	User		v	v	
13	Login			v	
14	Admin		v		v
15	AdminNews				v
16	AdminChange				v

The classes listed in table 9 are taken from 2 systems, namely the Old Portal system as a reference for the classes needed, and Learning IS as a reference for Information Systems which will become the new portal.

4. CONCLUSIONS

Based on the study results and detection of code smell towards 5 information systems that have been conducted, it can be concluded that there were 2 of 5 systems that were feasible to be developed through the refactor process. However, from the 2 systems, a system that used a framework was chosen to increase the productivity of system development [36], namely Learning IS. Metrics used to measure code smells could also be used to detect classes that were imposed code smells. The detection and filter processes of both the old system and the new candidate system obtained 16 classes that required special consideration in code refactor to develop the new portal system. The total number of class in the old IS and the new candidate was 29 classes. Thus, the percentage of detected code smells was 55.17%. This is quite high since more than half of the codes should be refactored, but it is quite important since the team didn't have to refactor the entire codes. This study also revealed the fact that code with a good structure would be easier to detect. The old portal which contained various systems was not orderly structured so that it could not be detected by code smell detection tools. Then according to the study objectives, it was also found that modular and structured codes could be read well by detection tools, and had a high LCOM rate compared to structured and native code.

ACKNOWLEDGEMENT

The research team would like to express gratitude to the Ministry of Research, Technology, and Higher Education for the funds granted to conduct this study with researc contract number SP DIPA042.06.1.401516/2021. The research team would also like to thank the Research and Community Service Department (LPPM) of Duta Wacana Christian University for the support provided during the study process so that it can be conducted and completed properly.

REFERENCES

1. R. Delima, H. B. Santoso and J. Purwadi, **Development of Dutatani Website Using Rapid Application Development**, International Journal of Information Technology and Electrical Engineering, vol. 1, no. 2, pp. 36-44, 2017.
2. A. R. Chrismanto, A. Wibowo, H. B. Santoso and R. Delima, **Developing Agriculture Land Mapping using Rapid Application Development (RAD): A Case Study from Indonesia**, International Journal of Advanced Computer Science and Applications, pp. 232-241, 2019.
3. R. Delima, F. Galih and A. Wibowo, **Development of Crop and Farmer Activity Information System**, Researchers World, vol. VIII, no. 4, pp. 180 - 189, October 2017.
4. R. Delima, H. B. Santoso, G. H. Aditya, J. Purwadi and A. Wibowo, **Development of Sales Modules for Agricultural E-Commerce Using Dynamic Syatem Development Method**, International Journal of New Media Technology (IJNMT), pp. 95-103, 2018.
5. R. Delima, H. B. Santoso, N. Andriyanto and A. Wibowo, **Development of Purchasing Module for Agriculture E-Commerce using Dynamic System Development**, International Journal of Advanced Computer Science and Application, pp. 86-96, 2018.
6. R. Delima, A. Wibowo, A. Rachmat Chrismanto and H. Budi Santoso, **A Model of Requirements Engineering on Agriculture Mobile Learning System Using Goal-Oriented Approach**, International Conference on Informatics and Computing (ICIC), Jakarta, 2020.
7. S. Charalampidou, A. Ampatzoglou, A. Chatzigeorgiou and P. Avgeriou, **Assessing Code Smell Interest Probability: A Case Study**, XP '17: Proceedings of the XP2017 Scientific Workshops, New York, 2017.
8. A. Kaur and M. Kaur, **Analysis of Code Refactoring Impact on Software Quality**, MATEC Web of Conferences ICAET, France, 2016.
9. M. Shariful Haque, J. Carver and T. Atkison, **Causes, Impacts, and Detection Approaches of Code Smell : A Survey**, ACMSE 2018 Conference, 2018.
10. F. M, B. K, B. J, O. W and R. D, **Refactoring: Improving the Design of Existing Code**, Boston: Addison-Wesley Professional, 1999.
11. E. Murphy-Hill, **Scalable, Expressive, and Context-Sensitive Code Smell Display**, OOPSLA 2008 ACM, Nashville, 2008.
12. J. Radatz, A. Geraci and F. Katki, **IEEE standard glossary of**, IEEE, 1990.
13. Nabilah and W. Danar Sunindyo, **Controlling Software Evolution Process Using Code Smell Visualization**, ICCCV, Jeju, 2019.
14. R. d. Mello, A. Uchoa, R. Oliveira and D. Oliveira, **Investigating the Social Representations of Code Smell Identification: A Preliminary Study**, International Workshop on Cooperative and Human Aspects of Software Engineering (CHASE), Montreal, 2019.
15. M. Platinni, M. Genero and L. Jimenez, **A Metric-Based Approach For Predicting Conceptual Data Models Maintainability**, International Journal of Software Engineering and Knowledge Engineering, vol. 11, no. 6, pp. 703-729, 2001.
16. J. Hauser and G. Katz, **Metrics: You are what you measure!**, European Management Journal, vol. 16, pp. 517-528, 1998.
17. F. A. Fontana, V. Ferme, A. Marino and B. Walter, **Investigating the Impact of Code Smells on System's Quality: An Empirical Study on Systems of Different Application Domains**, Software Maintenance (ICSM), 2013 29th IEEE International Conference, Eindhoven, 2013.
18. M. Lanza and R. Marinescu, **Object-Oriented Metrics in Practice**, Verlag Berlin Heidelberg: Springer, 2006.
19. Y. H. S. Al-Mamary, A. Shamsuddin and A. H. N. Aziati, **The Role of Different Types of Information Systems In Business Organizations : A Review**,

International Journal of Research (IJR), vol. 1, no. 7, pp. 1279-1286, 2014.

20. S. K. Boell and D. C. Kecmanovic, **What is an Information System?**, Hawaii International Conference on System Sciences, 2015.
21. D. A. Almazan, Y. S. Tovar and J. M. M. Quintero, **Influence of information systems on organizational results**, Contaduría y Administración, vol. 62, no. 2, pp. 321-338, 2017.
22. A. Arisman, **Analysis of Factors Affect to Organizational Performance In Using Accounting Information Systems Through Users Satisfaction and Integration Information Systems**, Sriwijaya International Journal Of Dynamic Economics And Business, vol. 1, no. 2, pp. 167-180, 2017.
23. K. M. Alhendawi and A. S. Baharudin, **The Assessment Of Information System Effectiveness In E-Learning, E-Commerce And Egovernment Contexts: A Critical Review Of The Literature**, Journal of Theoretical and Applied Information Technology, vol. 95, no. 18, pp. 4897-4912, 2017.
24. X. Zhang and J. L. Freschi, **Scalability analysis of three monitoring and information systems: MDS2, R-GMA, and Hawkeye**, Journal of Parallel and Distributed Computing, vol. 67, no. 8, pp. 883-902, 2007.
25. A. Chugh, S. Manchanda and P. Khosla, **Improving Software Maintainability Through Refactoring- An Empirical Study**, International Journal of Advances in Electronics and Computer Science, vol. 2, no. 4, pp. 88-93, 2015.
26. A. Wibowo, A. R. Chrismanto, H. B. Santoso dan R. Delima, **The Development of Mobile-based Farmland Mapping System with Drones and Wireless Devices**, International Journal of Advanced Trends in Computer Science and Engineering (IJATCSE), vol. 9, no. 5, pp. 7894-7902, 2020.
27. M. Fowler, **Refactoring: Improving the Design of Existing Code**, London: Pearson Education, 2018.
28. G. Lacerda, F. Petrillo, M. Pimenta and Y. G. Gueheneuc, **Code Smells and Refactoring: A Tertiary Systematic Review of Challenges and Observations**, Journal of Systems and Software, vol. 167, 2020.
29. J. P. d. Reis, F. B. e. Abreu, G. d. F. Carneiro and C. Anslow, **Code Smells Detection and Visualization: A Systematic Literature Review**, Archives of Computational Methods in Engineering, vol. 28, 2021.
30. T. Sharma, **A Survey on Software Smells**, Journal of Systems and Software, vol. 138, p. 50, 2017.
31. F. Palomba, G. Bavota, M. D. Penta, F. Fasano, R. Oliveto and A. D. Lucia, **A large-scale empirical study on the lifecycle of code smell co-occurrences**, Information and Software Technology, vol. 99, pp. 1-10, 2018.
32. A. Y. P. Putri, **Modifikasi Metode Function Point dengan Menambahkan Kompleksitas Proses Bisnis pada General System Characteristics untuk Estimasi Biaya Perangkat Lunak**, Institut Teknologi Sepuluh November, Surabaya, 2018.
33. V. Jain and A. Gupta, **Lack of Conceptual Cohesion of Methods: A new alternative to Lack Of Cohesion of Methods**, ISEC '15: Proceedings of the 8th India Software Engineering Conference, India, 2015.
34. R. Ponnala and C. Reddy, **Object Oriented Dynamic Metrics in Software Development: A Literature Review**, International Journal of Applied Engineering Research, vol. 14, no. 22, pp. 4161-4172, 2019.
35. B. L. Sousa, M. A. S. Bigonha and K. A. M. Ferreira, **When GOF Design Patterns occur with God Class and Long Method Bad Smells?-An Empirical Analysis**, Journal of Software Engineering Research and Development, vol. 17, pp. 11-22, 2019.
36. S. Syafiq, M. Daud, H. Hasan, A. Zairi, S. Imri, E. Akmar and N. Rahim, **Comparison of Web Development Using Framework over Library**, International Journal of Computer and Systems Engineering, vol. 12, no. 3, pp. 215-218, 2018.