A research on the effectiveness evaluation of aspect-based refactoring using code smells

Guan-Yu Chen, Kuo-Hsun Hsu
Department of Computer Science,
National Taichung University of Education,
Taichung,
Taiwan 403,
Email: c093128@gmail.com, glenn@mail.ntcu.edu.tw

ABSTRACT—Aspect-oriented refactoring is a modularization technique for reducing complexity of existing software systems through encapsulating crosscutting concerns. This technique makes it easier for developers to maintain and manage the system easier. Bad smells are problems that may occur in code or design phases and can be resolved by refactoring. However, bad smells may also appear in the system that is developed using Aspect-oriented refactoring.

This paper addresses the issues of evaluation of the Aspect-oriented refactoring effects through the use of a set of software metrics. By using a pre-defined set of interpretation rules to interpret the software metric results applied to Java source code, the software engineer can be provided with guidance as to the location of bad smells. By comparing the difference of the metrics before and after software refactoring, the aspect-oriented refactoring effects can be further studied and provide a better insight to the software development with aspect-oriented techniques.

Keywords: Aspect-oriented refactoring, measure, bad smells, effects.

I. INTRODUCTION

Aspect-orientation is an emerging paradigm to support the separation of concerns in software development [1]. This paradigm is aim to improve software quality by changing the internal code structure of an Object-Oriented system without affecting the overall behavior of the system.

There are numerous studies published in the literature on empirical evaluation of Aspect-oriented programming (AOP) paradigm [2]-[4]. In some of these studies, there have been numerous claims in favor and against AOP compared to OOP. Some results presented in these studies are subjective, some studies are inconclusive. Therefore, this paper propose a set of software quality evaluation method to obtain effect of aspect-oriented refactoring to OO system.

Theoretically, when an OO system goes through AO refactoring, bad smells will be removed to a certain extent. Usually, when an OOP system goes through AO refactoring, bad smells in the OO system will be removed to a certain extent. However, it is possible that AOP bad smells will be introduced in this process. Researches on bad smells often focus on either OOP systems or AOP systems, and the comparison of the effects of bad smells removal in the system before and after the refactoring is seldom addressed. Therefore, we propose, in this paper, a metric-based comparison scheme that compares the metrics generated from both OOP systems and AOP systems with aspect-oriented refactoring method. Considered. Through the comparison, it could give developers a better insight of AOP refactoring methods.

The rest of the paper is organized as follow: section II presents researches related to AO refactoring and AOP bad smells. The definition of the evaluation method of AO refactoring is described in section III. The conclusion and future work is given in section IV.

II. RELATED WORK

In the past decade, there are many researches proposed in the area of aspect-oriented programming. Most of the literatures described below focus on the quality metric of software in an AO system, and some of the literatures defined the definitions of AOP bad smell.

Software metrics are the means by which software quality can be evaluated. Ceccato, M and Tonella, P [5] proposed a based on a metrics suite that extends the metrics traditionally used with the OO paradigm for the cross-cutting degree of an aspect. The metrics includes a suite of coupling framework and the six coupling metrics. Zhao, J. [6] proposed several metrics for aspect cohesion based in aspect dependency graphs. They defined various coupling measures in terms of different types of dependencies between aspects and classes. Sant'Anna, Cláudio, et al. [7] developed a suite of metric which includes metrics adapted from known OOP metrics. The purpose of the framework is to increase understanding of separation of concerns, coupling, cohesion and size attributes as predictors of maintainability and reusability. Bartolomei, Thiago T., et al. [8] suggested a coupling framework for AspectJ, which is an extension of a coupling framework for OO systems. The extended framework contains a specific definition of the different coupling mechanisms found in AspectJ and an additional criterion instantiation. Zhao, Jianjun [9] propose a metric suite to quantify the information flows in an aspect-oriented program. These

metrics can be used to represent various dependence relations at different levels of an aspect-oriented program.

The code smell may occurred when the version of a system continuously updated. In AOP bad smell analysis method [10], they used an exploratory method to repeatedly observe the source codes among different versions, and analyze the code for bad smell in aspect oriented systems. The results show that their proposed smells occurred frequently. Srivisut et al. [11] defined AOP bad smells. They proposed an architecture to refactor the code that exists bad smells. Monteiro et al. [12] proposed the good programming style in aspect oriented programming. They redefined the OOP bad smells to make them suitable in developing aspect code and proposed new AOP bad smells and the example codes.

Most of these researches consider coupling or cohesion connection between classes and aspects in AO systems. However, the effect of AO refactoring of OO system into AO system is rarely discussed. For this reason, we will use software quality metric to identify OOP and AOP bad smells. Through the remove degree of bad smells to analyze the effect of AO refactoring.

III. METHODOLOGY

In this section, we refer the definitions and the identification methods of four OOP bad smells and six AOP bad smells from Mäntylä, Mika V., Casper Lassenius.[13] and Bertrán, Isela Macía [14].The OOP bad smells included large class, long method, long parameter list and duplicate code, and the AOP bad smell included god pointcut, forced join point, redundant pointcut, idle pointcut, god aspect and composition bloat. The system architecture of effective evaluation proposed in this rearch is shown in Figure 1. In this paper, the JHotDraw v.60b1 [16] is used to be our OO test, and the AJHotDraw v0.4 [17] is used to be our AO test. JHotDraw is a drawing open-source Java application. The 60b1 version has 486 classes, distributed across 30 packages, for 28,400 lines of Java source code. Corresponding with JhotDraw, AJHotDraw is an aspect-oriented version based on aspcetJ, an aspect language extending Java with crosscutting functionality. The 0.4 version has 291 classes, distributed across 18 packages, for 20,461 lines of aspcetJ source code.

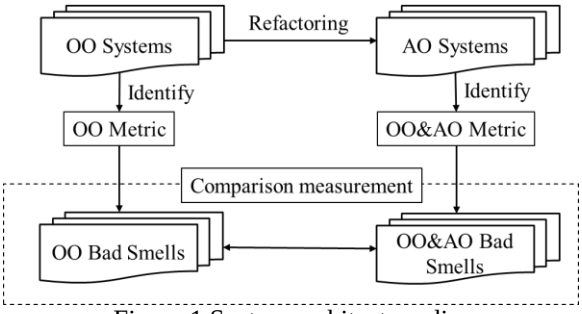

Figure 1 System architecture diagram

We will use software quality metrics in JHotDraw and AJHotDraw to get the measure value, and set a threshold value for each metric. Based on the measure value to identify OOP and AOP bad smells in JHotDraw and AJHotDraw. The identify results are compared with each other to verify the remove degree of bad smells. Through the remove degree of bad smells to analyze the effect of AO refactoring.

A. Identify OOP bad smells in JHotDraw

In this stage, we use Eclipse metrics plugin [15] in JHotDraw to get the necessary code metric. Eclipse metrics plugin measures various metrics calculation with average and standard deviation and type dependency analyzer plugin in the Eclipse platform. We analyze metric data from the Eclipse metrics plugin to identify OOP bad smells in JHotDraw system. The definition and identification methods of OOP bad smells is mainly adopted [13].

For example, the definition of large class is a class that contains many fields/ methods/ lines of code. Metrics related of large class are Number of attributes, Number of methods, Line of code, Cyclomatic Complexity and Lack of cohesion of methods. By collecting measurement of the above metrics of a software system, large class bad smell can be identified through a given set of detection rules. Figure 2 shows the measurement data.

Metric	Total	Mean	Std. Dev.	Maximum
Number of Attributes (avg/max per type)	20	4	7.51	19
Number of Methods (avg/max per type)	99	19.8	32.634	85
Total Lines of Code	688			
McCabe Cyclomatic Complexity (avg/max per method)		1.919	1.857	13
Lack of Cohesion of Methods (avg/max per type)		0.191	0.382	0.955

Figure 2 Measurement data of using OOP metric for JHotDraw (single class)

In order to detect bad smell in code, we defined the detection rules below. As a large class contains many fields/ methods/lines of code, we considered it as a large class when the following detection rule is satisfied for at least one of its advice:

Large class = (NOA > HIGH or NOM > HIGH) and (CC > HIGH or LOC > HIGH) or (LCOM > 1.0)

Based on the rule and the measurement, we selected one threshold for the number of attributes, we used the limits for a number of Large Class Attributes and methods. According to anti-pattern book [18] that a class with more than 60 attributes and operations usually indicates the presence of the large class. For the number of attributes (NOA), we used limits of 10 attributes, and for the number of methods (NOM), the limits is 50. According to the Java code conventions [19], files longer than 1500 lines are cumbersome and should be avoided. However, this code conventions is not talking about lines of code. Lines of file in general, which includes comments, code outside methods. Therefore, we used threshold of 500 lines of code

(LOC). According to McCabe [20] proposed the threshold of a program cyclomatic complexity (CC) should be 10, we follow this suggestion. Lack of cohesion of methods (LCOM) is a measure for the cohesiveness of a class. Calculated with Henderson-Sellers method [22], this metric is normalized to a real number between 0.0 and 2.0. If the metric value over 1.0 being viewed as being suggestive of poor design.

In figure 2, the total number of attributes is 20 and total number of method is 99, these two metrics value has exceeded the threshold. The first part of the detection rule is satisfied. The total number of LOC is 688, and maximum value of CC is 13, these two metrics value has exceeded the threshold. The second part of the detection rule is also satisfied. Although the metric value of LCOM is less than 1.0, this class may still be recognized as a class with large class bad smell.

B. Identify OOP & AOP bad smells in AJHotDraw

In this stage, we use OOP metrics and AOP metrics in AJHotDraw to identify OOP bad smells and AOP bad smells. The identify results are compared with the those in the previous stage to verify the remove degree of bad smells.

According to [14], we develop an AOP metric system. For example, the definition of god ponitcut is a kind of bad smell with complicated description or too many concerns in the pointcut. The related metrics is Set of primitive pointcuts referring a pointcut, Set of join points, Cyclomatic complexity and Line of code. Figure3 and Table 1 show the metrics results of using OOP metrics and AOP metrics for AJHotDraw. Based on these metrics, god pointcut bad smell van be analyzed and detected.

Metric	Total	Mean	Std. Dev.	Maximum
Number of Attributes (avg/max per type)	19	3.8	7.111	18
Number of Methods (avg/max per type)	98	19.6	32.234	84
Total Lines of Code	667			
McCabe Cyclomatic Complexity (avg/max per method)		1.888	1.862	13
Lack of Cohesion of Methods (avg/max per type)		0.191	0.382	0.955

Figure 3 Measurement data of using OOP metric for AJHotDraw (single class)

Table 1 Measurement data of using AOP metric for JHotDraw (single class)

Metric	Value
Set of primitive pointcuts referring a pointcut	14
Set of join points	34
Cyclomatic complexity	1
Line of Code	17

Figure 3 is result of using OOP metrics for AJHotDraw. Compare with measurement data of Figure 2, we discover the value of NOA, NOM and LOC is slightly reduced, and the value of CC, LCOM is unchanged. Although there are three metrics value decreasing, the metrics value still exceed the threshold value, which means that this class still has large class bad smell and the AOP refactoring does not provide a better refactoring result in this case.

We adopt identification method in [14] to identify AOP bad smells in AJHotDraw. The detection rule is as follows:

The metric SPP (Set of Primitive Pointcuts refer-

God Pointcut = (SPP > HIGH or SJP > HIGH) and (CC > HIGH or LOC > VERY_HIGH)

ring a pointcut) counts the number of primitive pointcuts (e.g. call, execution) used in the pointcut expression. The metric SJP (Set of Join Points) computes the Set of join point shadows captured by a given pointcut. For the metric of SPP, we use limits of 10 keywords, and for SJP, the limits of 10 join points. The metric CC (Cyclomatic Complexity) corresponds to the complexity of the advice that referring a given pointcut. Finally, the metric LOC (Line of Code) counts the number of lines of code involving in the advice implementation. For the metric of CC, we use limits of 4, and for LOC, the limits of 20 lines.

In table 1, the total number of SPP is 14, and total number of SJP is 34, these two metrics value has exceeded the threshold. The first portion of the detection rule is satisfied. The maximum Number of CC is 1 and the maximum number of LOC is 17, which does not satisfy the second portion of the rule. Since only one portion of rule is satisfied, this class is not considered to contain the god pointcut bad smell.

IV. CONCLUSION AND FUTURE WORK

In this paper, an issue to the discovery of bad smells in OOP systems and AOP systems are discussed according to the definition, we proposed to evaluate the effects of AOP refactoring by comparing the various metrics for identifying bad smells before and after the refactoring, which provide developers a better insight of choosing refactoring method in this study, aspect-oriented refactoring.

Our future work includes improve the AOP metric system, making it to be able to identify more AOP bad smells, and trying to find out the possible links relationship between OOP bad smells and AOP bad smells.

REFERENCES

[1] Bergmans, Lodewijk, and Mehmet Aksit. "Composing crosscutting concerns using composition filters." *Communications of the ACM* 44.10 (2001): 51-57.

[2] Sirbi, Kotrappa, and Prakash Jayanth Kulkarni. "Metrics for aspect oriented programming-an empirical study." *International Journal of Computer Applications* 5.12 (2010).

[3] Macia Bertran, Isela, Alessandro Garcia, and Arndt von Staa. "An exploratory study of code smells in evolving aspect-oriented systems." *Proceedings of the tenth international conference on Aspect-oriented*

software development. ACM, 2011.
[4] Franca, Joyce, et al. "An Empirical Evaluation of Refactoring Crosscutting Concerns into Aspects Using Software Metrics." *Information Technology: New Generations (ITNG), 2013 Tenth International Conference on*. IEEE, 2013.
[5] Ceccato, Mariano, and Paolo Tonella. "Measuring the effects of software aspectization." *1st Workshop on Aspect Reverse Engineering*. Vol. 12. 2004.
[6] Zhao, Jianjun. "Measuring coupling in aspect-oriented systems." 10th International Software Metrics Symposium (Metrics 04). 2004.
[7] Sant'Anna, Cláudio, et al. "On the reuse and maintenance of aspect-oriented software: An assessment framework." *Proceedings of Brazilian symposium on software engineering*. 2003.
[8] Bartolomei, Thiago T., et al. "Towards a unified coupling framework for measuring aspect-oriented programs." *Proceedings of the 3rd international workshop on Software quality assurance*. ACM, 2006.
[9] Zhao, Jianjun. "Towards a metrics suite for aspect-oriented software." Rapport technique (2002).
[10] Macia Bertran, Isela, Alessandro Garcia, and Arndt von Staa. "An exploratory study of code smells in evolving aspect-oriented systems." *Proceedings of the tenth international conference on Aspect-oriented software development*. ACM, 2011.
[11] Macia Bertran, Isela, Alessandro Garcia, and Arndt von Staa. "An exploratory study of code smells in evolving aspect-oriented systems." *Proceedings of the tenth international conference on Aspect-oriented software development*. ACM, 2011.
[12] Monteiro, Miguel P., and João M. Fernandes. "Towards a catalogue of refactorings and code smells for aspectj." *Transactions on aspect-oriented software development I*. Springer Berlin Heidelberg, 2006. 214-258.
[13] Mäntylä, Mika V., and Casper Lassenius. "Subjective evaluation of software evolvability using code smells: An empirical study." *Empirical Software Engineering* 11.3 (2006): 395-431.
[14] Macia, I. "On the detection of architecturally-relevant code anomalies in software systems." *Phd, DI, PUC-Rio* (2013).
[15] Sauer, F. "Eclipse metrics plugin 1.3.6." *URL metrics.sourceforge.net* (2005).
[16] Gamma, Erich, and Thomas Eggenschwiler. "JHotDraw." (2004).
[17] Marin, Marius. "Ajhotdraw." (2007).
[18] Akroyd, Michael, "AntiPatterns Session Notes," Object World West, San Francisco, 1996.
[19] Type-Safe, P. H. P. "Java Code Conventions." (2014).
[20] McCabe, Thomas J. "A complexity measure." *Software Engineering, IEEE Transactions on* 4 (1976): 308-320.
[21] Watson, Arthur H., Thomas J. McCabe, and Dolores R. Wallace. "Structured testing: A testing methodology using the cyclomatic complexity metric." *NIST special Publication* 500.235 (1996): 1-114.
[22] Henderson-Sellers, Brian. *Object-oriented metrics: measures of complexity*. Prentice-Hall, Inc., 1995