2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering

Do Developers Focus on Severe Code Smells?

Tsubasa Saika*, Eunjong Choi*, Norihiro Yoshida†, Shusuke Haruna* and Katsuro Inoue*
*Osaka University, Japan
{t-saika@ist, ejchoi@osipp, haruna@ist, inoue@ist}.osaka-u.ac.jp
†Nagoya University, Japan
yoshida@ertl.jp

***Abstract*—Code smells are structures in the code that suggest the possibility of refactoring. To prioritize code smells in large-scale source code, several tools for refactoring calculate their severity based on software metrics. Although several metrics are known as maintainability indicators, it is still unclear whether these severity indicators are in line with developer's perception. In this paper, we investigate whether developers focus on severe code smells. The result shows that developers focus on only particular types of severe smells and refactoring do not decrease the severity of code smell significantly.**

I. Introduction

Developers tend to write source code with low maintainability even when they can easily avoid it. (e.g., because they are in a hurry or implementing urgent patches or simply making suboptimal choices) [1].

Code smells (also known as bad smells) are symptoms of poor design and implementation choices that may hinder code comprehension and possibly increase change and fault-proneness. They are also used to find structures in the code that suggest the possibility of refactoring [2]. For example, *Blob Class* [2] is a large and complex class that centralizes the behavior of a portion of a system and only uses other classes as data holders. It can rapidly grow out of control, making it harder to fix bugs and add new features [1].

Many code quality analysis tools can detect code smell in source code automatically (e.g., inFusion, PMD). Since a number of code smells are detected by these tools in large-scale source code, developers must determine which code smells should be preferentially removed [3]. However, each code smell indicates only the location and the kind of symptoms but not the the severity.

To prioritize code smells in large-scale source code, several tools (e.g., inFusion, inCode) calculate their severity using software metrics as quantitative indicators of software maintainability. Although several metrics are used for not only maintainability indicators [4], [5] but also the identification of refactoring opportunities [6], it is still unclear whether these severity indicators are in line with developer' perception. If the severity is not in line with developers' perception, severity-based prioritization is inappropriate to support refactoring.

In this study, we investigate the refactorings that were performed by developers in three mature and large-scale OSS systems. If developers preferentially performed refactoring on code with high severity in the OSS systems, the severity-based prioritization is a useful approach to finding refactoring opportunities. Also, if developers decreased the severity of code smells by refactoring in the OSS systems, the severity is in line with developers' perception of maintainability.

We answer the following research questions :

- RQ1 Do developers perform refactoring more frequently on code with more sever code smell?
- RQ2 Does refactoring decrease the severity of code smells?

Our findings from the RQs are as follows:

- Developers focus on particular types of severe smells. Therefore, the severity-based prioritization is a useful approach to find the paticular type of refactoring opportunities.
- Refactoring do not decrease the severity of code smell significantly. Therefore, the severity is not in line with developer's perception of the maintainability.

II. Datasets of Refactorings and Code Smells

In this study, we analyzed datasets of refactorings and code smells that were detected in 64 releases of three Java OSS systems: **Apache Ant (Ant)**[1], **ArgoUML (Argo)**[2], and **Xerces-J (Xerces)**[3]. Table I shows statistical data on each of the systems. In this table, the last row shows the overall number of three systems. In the following subsections, we explain the dataset of refactorings (Section II-A) and the extracted code smells (Section II-B).

A. Dataset of Refactorings

We used the dataset of refactorings that was collected in a previous study by Bavota et al. [7]. This dataset contains refactorings that were detected by Ref-finder [8], a templated-based refactoring detection tool. Each refactoring was manually validated to exclude false positives. From the dataset, we can observe distinct types, modified class names, and release version of each refactoring. We selected this dataset because its refactorings are reliable.

Table II depicts the total number of analyzed refactorings and the number of distinct types of refactoring in the dataset.

[1]http://ant.apache.org/
[2]http://argouml.tigris.org/
[3]http://xerces.apache.org/xerces-j/

978-1-5090-1855-0/16 $31.00 © 2016 IEEE
DOI 10.1109/SANER.2016.117

TABLE I
STATISTICS DATA OF ANALYZED SYSTEMS

System	Period	Analyzed	# Releases	# Classes
Xerces	Oct. 2003 - Nov. 2006	1.0.0-2.9.0	34	19,567
Argo	Oct. 2002 - Apr. 2011	0.10.1-0.32.2	12	43,686
Ant	Aug. 2003 - Dec. 2010	1.1-1.8.2	18	22,768
Overall	-	-	64	86,021

TABLE II
OVERVIEW OF REFACTORING DATASET

System	# Refactorings	# Distinct Types
Xerces	7,502	24
Argo	3,255	21
Ant	1,289	16
Overall	12,043	28

TABLE III
NUMBERS OF DETECTED CODE SMELLS

Type of Code Smell	Xerces	Argo	Ant	All
Blob Class	665	83	87	835
Data Class	71	3	28	102
Distorted Hierarchy	9	0	0	9
God Class	952	160	143	1,255
Refused Parent Bequest	114	14	81	209
Schizophrenic Class	39	48	4	91
Tradition Breaker	99	3	0	102
Classes w/ smells	1,949	311	343	2,603
Classes w/o smells	17,618	43,375	22,425	83,418
Overall classes	19,567	43,686	22,768	86,021

B. Dataset of Code Smells

For the detection of code smells, we used inFusion[4], a tool that detects 24 types of code smells based on code metrics [5]. To detect each type of code smell, inFusion uses pre-defined conditional expression consisting of logical combination of code metrics with different threshold values. For example, *Schizophrenic Class* is detected based on two metrics named *Number of Public Methods* and *Tight Capsule Cohesion*. It outputs detected code smells and their severity score on a scale of 1 to 10 based on how much relative metrics values exceed the threshold value. We adopted inFusion because it has ability to detect various types of code smells as well as to measure their severity.

From the detected code smells, we chose class as level of granularity because refactoring dataset only provides a class name with each refactoring. Table III shows the number of class-level code smells detected in the systems. In this table, the last row shows the overall numbers of code smells in the systems. As you can see in this table, only 7 class-level code smells were detected. Because multiple code smell can be detected in a single class, the overall number of classes might contain more than one code smell.

[4]http://www.intooitus.com/products/infusion

[5]The details of the 24 types of code smells can be found at https://www.intooitus.com/products/infusion/detected-flaws

TABLE IV
RESULTS OF MANN-WHITNEY U-TEST FOR RQ1

	RQ1	RQ2
Blob Class	✓	✓
Data Class	n/a	n/a
Distorted Hierarchy		n/a
God Class	✓	✓
Refused Parent Bequest		
Schizophrenic Class	✓	
Tradition Breaker	✓	

III. INVESTIGATION AND RESULTS

This section details our investigation and results, aimed at addressing the RQ1 and RQ2. This section also identifies threats to validity of our results.

A. Do developers perform refactoring more frequently on code with more severe code smell?

To answer RQ1, we investigated the relationship between refactorings and the severity of code smells. We used the **Mann-Whitney U test** [9], a nonparametric significance test to find out whether refactored classes have more severe code smells than non-refactored classes. The Mann-Whitney U test was used to investigate the statistical significant of the differences between two independent groups: refactored and non-refactored classes that have at least one code smell. The dependent variable is the severity of each code smell, which is ordinal variable measured from 1 to 10.

The results of the Mann-Whitney U test can be seen in Table IV. In this table, the check marks represent the existence of significant differences ($p > .05$) and 'n/a' indicates that the Mann-Whitney U test was not applied. As we can see in the table, there are significant differences between refactored classes and non-refactored classes for *Blob Class*, *God Class*, *Schizophrenic Class* and *Tradition Breaker*. This implies that developers tend to perform refactoring more frequently on code with more severe *Blob Class*, *God Class*, *Schizophrenic Class* and *Tradition Breaker*.

Summary for RQ1: For *Blob Class*, *God Class*, *Schizophrenic Class* and *Tradition Breaker*, there are statistical tendencies for developers to perform refactoring more frequently on code with more severe code smell.

B. Does refactoring decrease the severity of code smells?

To answer RQ2, we analyzed refactorings and the changes of the severity of code smell from the dataset. We conducted the Mann-Whitney U test to reveal whether there are significant differences between the severity of refactored classes

and non-refactored classes. The Mann-Whitney U test was only applied to classes that have at least one code smell. The dependent variable is the change of the severity of each code smell through release versions which are ordinal variables measured from -10 to 10. If a refactoring largely increases or decreases the severity of code smells, the test must show significant differences.

The results of the Mann-Whitney U test can be seen in Table IV. In this table, the check marks represent the existence of significant differences ($p > .05$) and 'n/a' indicates that the Mann-Whitney U test was not applied because they are classes without code smells or non-refactored classes. From this table, we can observe significant difference only from *Blob Class* and *God Class*. This implies that in most case, the effect of refactoring on the changes of severity of code smell are relatively small. Especially for *Blob Class* and *God Class*, the severity of code smells were increased as well as decreased, regardless of performance of refactoring. From this results, we can conclude that refactoring does not decrease the severity of code smells.

Summary for RQ2: A comparison of refactored and non-refactored classes shows that refactoring does not statistically decrease the severity of code smells.

C. Threats to Validity

Some threats need to be considered when interpreting our study results. The first limitation was that the investigation results might have been too dependent on the dataset and output of the inFusion. However, the dataset were validated in the [7] and inFusion is a commercial code smell detection tool . Therefore, we believe that the results of investigation in this study are reliable.

Moreover, investigating different systems could have led to different results because our case study was conducted on three Java OSS systems. However, we believe that our investigation results can be generalized an applied to other open source software systems because they spanned 64 release versions from three separate systems.

IV. SUMMARY

To reveal whether developers focus on severe code smell, this study investigated the effect of the severity of code smells on refactorings in 64 releases of three Java OSS systems.

Based on our investigation results, it turns out that developers perform refactoring more frequently on code with more severe code smell for *Blob Class*, *God Class*, *Schizophrenic Class* and *Tradition Breaker*. Therefore, it is considered that developers can use these severity of code smells as good indicators for performing refactoring. It also revealed that refactorings do not decrease the severity of code smells significantly.

As future work, we plan to analyze additional software systems to achieve the generality of our findings. In addition, we would like to extend our investigation for more types of code smells such as method-level code smells.

REFERENCES

[1] F. Palomba, G. Bavota, M. Di Penta, R. Oliveto, D. Poshyvanyk, and A. De Lucia, "Mining version histories for detecting code smells," *IEEE Transactions on Software Engineering*, vol. 41, no. 5, pp. 462–489, 2015.

[2] M. Fowler, *Refactoring: Improving the Design of Existing Code*. Addison Wesley, 1999.

[3] A. Yamashita and L. Moonen., "Exploring the impact of inter-smell relations on software maintainability: An empirical study," in *Proc. of ICSE*, 2013, pp. 682–691.

[4] B. Henderson-Sellers, *Object-Oriented Metrics: Measures of Complexity*. Prentice Hall, 1996.

[5] R. Marinescu and D. Ratiu, "Quantifying the quality of object-oriented design: the factor-strategy model," in *Proc. of WCRE*, 2004, pp. 192–201.

[6] F. Simon, F. Steinbruckner, and C. Lewerentz, "Metrics based refactoring," in *Proc. of CSMR*, 2001, pp. 30–38.

[7] G. Bavota, B. D. Carluccio, A. D. Lucia, M. D. Penta, R. Oliveto, and O. Strollo, "When does a refactoring induce bugs? an empirical study," in *Proc. of SCAM*, 2012, pp. 104–113.

[8] M. Kim, M. Gee, A. Loh, and N. Rachatasumrit, "Ref-Finder: a refactoring reconstruction tool based on logic query templates," in *Proc. of FSE*, 2010, pp. 371–372.

[9] W. J. Conover, *Practical nonparametric statistics*. John Wiley & Sons, 1971.