Revisiting the Relationship Between Code Smells and Refactoring

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

***Abstract*—Refactoring is a critical technique in evolving software systems. Martin Fowler presented a catalogue of refactoring patterns that defines a list of code smells and their corresponding refactoring patterns. This list aimed at supporting programmers in finding suitable refactoring patterns that remove code smells from their systems. However, a recent empirical study by Bavota et al. shows that refactoring rarely removes code smells which do not align with Fowler's catalog. To bridge the gap between them, we revisit the relationship between code smells and refactorings. In this study, we investigate whether developers apply appropriate refactoring patterns to fix code smells in three open source software systems.**

I. Introduction

Much research have been done to empirically investigate the role of refactoring and its impact on software metrics [1], [2]. Kim et al. investigated the role of API-level refactoring using their refactoring detection technique [3] and conducted a field study of refactoring challenges and benefits at Microsoft [4], [5]. Bavota et al. investigated the impact of refactoring on bugs in source code [6].

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 used to find structures in the code that suggest the possibility of refactoring [7]. For example, *Blob Class* [7] is a large and complex class that centralizes the behavior of a portion of a system and mainly uses other classes as data holders. It can rapidly grow out of control, making it harder to fix bugs and to add new features [8]. The impact of code smells has been also studied by many researchers. Yamashita and her colleagues investigated the capability of code smell detection for predicting and explaining maintenance problems [9], [10], the impact of inter-smell relations on code maintainability [11], [12], and the extent to which code smells reflect factors affecting maintainability that have been identified as important by programmers [13]. Sjøberg et al. quantified the effect of code smells on maintenance effort [14]. The two surveys [15], [16] have been conducted on programmer's perception of code smells. Tufano et al. investigated when and how code starts to smell bad [17].

So far, it is generally considered that a code smell should be removed once a refactoring pattern is applied to it [7], [18]. However, recent empirical studies show that code smells were rarely removed even if refactoring pattern was applied on code entities affected by the code smell [19], [20]. Bavota et al. investigated the extent to whether refactorings were applied to classes exhibiting code smells and able to remove code smells. The result shows that 42% of refactoring operations were performed on code entities affected by code smells, and only 7% of the performed operations actually removed the code smells from the affected class. Also, Saika et al. investigated the impact of the severity of code smell on refactoring [20]. The result shows that refactoring did not decrease the severity of code smells significantly.

To bridge the gap between the common perception of refactoring and the empirical studies, we revisit the relationship between code smell and refactoring. At first, we make a list of the corresponding refactoring patterns for each type of code smells according to not only books written by Fowler and Lanza et al. [7], [21] but also refactoring instances in three OSS systems by developers. And then, we investigate whether corresponding refactoring patterns are applied to code smells in the three OSS systems.

The main contributions of this study are as follows:

- We made a list of the corresponding refactoring patterns for each type of code smell.
- We revealed that applied refactoring patterns rarely correspond to the type of code smell in most cases. This result suggests one of the considerable reasons for the empirical result [19] indicating a code smell was rarely removed even if a refactoring pattern was applied on code entities affected by the code smell.

II. Datasets Analyzed

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 the following paragraphs, we explain the dataset of refactorings and the extracted code smells.

We used an existing dataset of refactorings that was collected in a previous study by Bavota et al. [6]. This dataset

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

TABLE I
STATISTICS DATA OF ANALYZED SYSTEMS

System	Analyzed	# Releases	# Classes
Xerces	1.0.0-2.9.0	34	19,567
Argo	0.10.1-0.32.2	12	43,686
Ant	1.1-1.8.2	18	22,768
Overall	-	64	86,021

TABLE II
AN OVERVIEW OF REFACTORING DATASET

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

contains refactorings that were detected by Ref-finder [22], a templated-based refactoring detection tool. Each refactoring was manually validated to exclude false positives by Bavota et al. From the dataset, we can observe distinct types, modified class names, and release version of each refactoring. Table II depicts the total number of analyzed refactorings and the number of distinct types of refactoring in the dataset.

For the detection of code smells, we used a tool inFusion[4] that detects 24 common types of code smells[5] because the number of the types that can be detected by inFusion is larger than the empirical study that was performed by Bavota et al. [19]. Table III shows the number of code smells detected in the systems. 16 out of 24 types of code smells were detected from the systems.

III. LIST OF CORRESPONDING REFACTORINGS

Tables IV and V show the lists of corresponding refactoring patterns for class-level and method-level code smells, respectively. To prevent missing of corresponding patterns, we referred not only the two popular books that are written by Fowler and Lanza et al. but also refactoring instances in the dataset. The approach for making these lists consist of the following two steps:

1) **Book-based identification**
   Identify the lists of the corresponding refactoring patterns using two popular books that are written by Fowler and/or Lanza et al. [7], [21].
2) **Instance-based identification**
   (i) For each refactoring instance in the dataset, identify the name of the applied refactoring pattern and the type of the code smell. (ii) And then, validate manually whether the refactoring instance alleviated or removed the code smell[6]. (iii) Finally, add the name of the refactoring pattern into the list of the corresponding refactoring patterns for the type of the code smell if at least one refactoring instance in the dataset alleviated or removed the code smell using the refactoring pattern.

[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

[6]Please note that this validation was performed by the second author of this paper.

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
Blob Operation	2,428	545	413	3,386
Data Clumps	834	434	54	1,322
External Duplication	713	269	46	1,028
Feature Envy	723	110	220	1,053
Intensive Coupling	476	463	132	1,071
Internal Duplication	701	160	85	946
Message Chains	19	9	1	29
Shotgun Surgery	39	10	34	83
Sibling Duplication	927	545	131	1,603

In Tables IV and V, each refactoring type in the second column denotes possible refactoring pattern to contribute in the alleviation or the removal of such of code smell in the first column of the same row. Pattern names listed in boldface were added in Step 1 based on their effectiveness in fixing the type of code smells according to Fowler's and/or Lanza's books [7], [21], respectively. The rest of refactoring pattern names were added in Step 2.

IV. INVESTIGATION

In this section, we automatically investigate whether corresponding refactoring patterns are applied to code smells in three open source software systems. In this investigation, we automatically executed the following two steps for each refactoring instance in the dataset of refactoring.

A) For each refactoring instance in the dataset of refactoring, we determine whether refactoring is performed on code entities affected by a code smell.
B) If the answer of Step A is yes, we determine whether a corresponding refactoring pattern is applied to the code smell.

Tables VI and VII show the results of this investigation. In most types of code smells, corresponding refactoring patterns were rarely applied to code smells. Among refactored classes affected by class-level code smell, only 26.8% of classes were refactored by corresponding patterns. In the case of method-level code smells, only 16.0% of refactored classes were refactored by corresponding patterns. Only *Blob classes* and *Blob operation* were frequently refactored by corresponding patterns (85.7% and 68.8% respectively). The other types of smells were rarely refactored by corresponding patterns.

The investigation result suggests a considerable answer for the empirical study [19] that shows that code smells were rarely removed even if refactoring pattern was applied on code entities affected by the code smell. The answer based on the investigation result is as follows:

TABLE IV
CLASS-LEVEL CODE SMELLS AND CORRESPONDING REFACTORING (PATTERNS IN BOLDFACE WERE IDENTIFIED FROM THE BOOKS WRITTEN BY FOWLER [7] AND/OR LANZA ET AL. [21])

Code Smells	Refactorings
Blob Class	**Extract Interface**, **Extract Subclass**, **Extract Class**, **Replace Data with Object**, Consolidate Cond Expression, Consolidate Duplicate Cond Fragments, **Extract Method**, Extract Superclass, Introduce Parameter Object, Move Method, Preserve Whole Object, Pull-up Field, Pull-up Method, Remove Control Flag, Replace Conditional with Polymorphism, Replace Method with Method Object, Replace Nested Cond Guard Clauses, Separate Query from Modifier
Data Class	**Move Method**, **Encapsulate Field**, **Encapsulate Collection**
Distorted Hierarchy	No corresponding pattern is found.
God Class	**Extract Interface**, **Extract Subclass**, **Extract Class**, **Replace Data with Object**, Extract Hierarchy, Extract Superclass, Move Field, Move Method, Pull-up Field, Pull-up Method, Replace Conditional with Polymorphism
Refused Parent Bequest	**Replase Inheritence With Delegation**, Change Bi to Uni, Move Field, Move Method, **Extract Class**
Schizophrenic Class	Extract Interface, Extract Superclass, Move Method
Tradition Breaker	Move Field, Move Method, **Extract Class**, **Pull-up Method**, **Pull-up Field**

TABLE V
METHOD-LEVEL CODE SMELLS AND CORRESPONDING REFACTORING PATTERNS (PATTERNS IN BOLDFACE WERE IDENTIFIED FROM THE BOOKS WRITTEN BY FOWLER [7] AND/OR LANZA ET AL. [21])

Code Smells	Refactorings
Blob Operation	**Extract Method**,**Replace Method with Method Object**, **Replace Temp with Query**, **Decompose Conditional**, Consolidate Cond Expression, Consolidate Duplicate Cond Fragments, Extract Hierarchy, Introduce Parameter Object, Preserve Whole Object, Remove Control Flag, Replace Conditional with Polymorphism, Replace Nested Cond Guard Clauses
Data Clumps	**Extract Class**, **Introduce Parameter Object**, **Preserve Whole Object**
External Duplication	**Extract Method**, **Extract Class**, **Pull-up Method**, **Form Template Method**, Replace Method with Method Object
Feature Envy	**Extract Method**, **Move Method**, **Pull-up Method**, **Move Field**
Intensive Coupling	Extract Method, Inline Method
Internal Duplication	**Extract Method**, **Extract Class**, **Pull-up Method**, **Form Template Method**, Consolidate Cond Expression, Move Method, Replace Method with Method Object
Message Chains	**Hide Delegate**
Shotgun Surgery	**Move Method**, **Move Field**, **Inline Class**
Sibling Duplication	**Extract Method**, **Extract Class**, **Pull-up Method**, **Form Template Method**, Consolidate Cond Expression, Replace Method with Method Object

TABLE VI
RESULT OF CLASS-LEVEL CODE SMELLS

Code Smells	# Classes	# Refactored Classes	# Classes w/ Corresponding Refactorings	Ratio
Blob Class	692	210	180	85.7%
Data Class	1,679	43	8	18.6%
Distorted Hierarchy	8	4	0	0.0%
God Class	1,294	284	54	19.0%
Refused Parent Bequest	538	40	7	17.5%
Schizophrenic Class	311	34	9	26.5%
Tradition Breaker	249	15	3	20.0%
Total	4,771	630	261	26.8%

Since corresponding patterns are rarely applied to code smells, refactoring rarely removes the code smells. In the cases of only *Blob Class* and *Blob Operation*, refactoring frequently remove those types of smells because many refactoring patterns can be used for the alleviation or the removal of those types of smells.

According to the investigation result, researchers should identify corresponding refactoring patterns for each smell, and then perform an empirical study for better understanding of the relationship between refactoring and code smells.

V. THREATS TO VALIDITY

We identify two main threats to validity.

We used the dataset of refactoring as described in Section II. Refactoring instances in the dataset were detected by a refactoring detection tool Ref-finder [22] but were manually validated by Bavota et al. [6]. This dataset is available online[7]. As future work, we need to replicate this study using other refactoring detection tools (e.g., JDevAn [23], Ref-detector [24]).

We used *inFusion* to detect code smell from each release version of source code. To our knowledge, it is a state-of-

[7]https://dibt.unimol.it/reports/refactoring-defect

TABLE VII
RESULT OF METHOD-LEVEL CODE SMELLS

Code Smells	#Methods	#Refactored Methods	#Methods w/ Corresponding Refactorings	Ratio
Blob Operation	4,610	436	300	68.8%
Data Clumps	1,402	50	0	0.0%
External Duplication	1,656	69	3	4.3%
Feature Envy	1,364	86	9	10.5%
Intensive Coupling	1,290	107	24	22.4%
Internal Duplication	2,218	164	28	17.1%
Message Chains	31	4	0	0.0%
Shotgun Surgery	65	3	0	0.0%
Sibling Duplication	1,956	147	31	21.1%
Total	14,592	1,066	395	16.0%

the-art commercial tool for detecting 24 types of code smells. Please note that it is larger number than 11 types in Bavota's empirical study [19]. The dataset of code smells is available at https://sites.google.com/site/yoshidaatnu/home/dataset. As future work, we plan to replicate this study using other approaches for detecting code smells [8] and other datasets [19].

VI. SUMMARY

In this paper, we revisited the relationship between code smell and refactoring. At first, we identified a list of the corresponding refactoring patterns for each type of code smell, and then investigate whether corresponding refactoring patterns are applied to code smells in three open source software systems.

The investigation result suggests a considerable answer for the empirical study [19]. The answer based on the investigation result is that refactoring rarely removes the code smell because corresponding pattern is rarely applied to a code smell.

ACKNOWLEDGMENT

This work was supported by JSPS KAKENHI Grant Numbers 25220003, 26730036 and 15H06344.

REFERENCES

[1] K. Stroggylos and D. Spinellis, "Refactoring–does it improve software quality?" in *Proc. of WoSQ*, 2007.

[2] O. Chaparro, G. Bavota, A. Marcus, and M. Di Penta, "On the impact of refactoring operations on code quality metrics," in *Proc. of ICSME*, 2014, pp. 456–460.

[3] M. Kim, D. Cai, and S. Kim, "An empirical investigation into the role of api-level refactorings during software evolution," in *Proc. of ICSE*, 2011, pp. 151–160.

[4] M. Kim, T. Zimmermann, and N. Nagappan, "A field study of refactoring challenges and benefits," in *Proc. of FSE*, 2012, pp. 50:1–50:11.

[5] ——, "An empirical study of refactoringchallenges and benefits at microsoft," *IEEE Trans. Softw. Eng.*, vol. 40, no. 7, pp. 633–649, 2014.

[6] 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.

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

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

[9] A. Yamashita and L. Moonen, "To what extent can maintenance problems be predicted by code smell detection? – an empirical study," *Information and Software Technology*, vol. 55, no. 12, pp. 2223–2242, 2013.

[10] A. Yamashita, "Assessing the capability of code smells to explain maintenance problems: an empirical study combining quantitative and qualitative data," *Empirical Software Engineering*, vol. 19, no. 4, pp. 1111–1143, 2013.

[11] 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.

[12] A. Yamashita, M. Zanoni, F. Fontana, and B. Walter, "Inter-smell relations in industrial and open source systems: A replication and comparative analysis," in *Proc. of ICSME*, 2015, pp. 121–130.

[13] A. Yamashita and L. Moonen, "Do code smells reflect important maintainability aspects?" in *Proc. of ICSM*, 2012, pp. 306–315.

[14] D. Sjøberg, A. Yamashita, B. Anda, A. Mockus, and T. Dyba, "Quantifying the effect of code smells on maintenance effort," *IEEE Trans. Softw. Eng.*, vol. 39, no. 8, pp. 1144–1156, 2013.

[15] A. Yamashita and L. Moonen, "Do developers care about code smells? an exploratory survey," in *Proc. of WCRE*, 2013, pp. 242–251.

[16] F. Palomba, G. Bavota, M. Di Penta, R. Oliveto, and A. De Lucia, "Do they really smell bad? a study on developers' perception of bad code smells," in *Proc. of ICSME*, 2014, pp. 101–110.

[17] M. Tufano, F. Palomba, G. Bavota, R. Oliveto, M. Di Penta, A. De Lucia, and D. Poshyvanyk, "When and why your code starts to smell bad," in *Proc. of ICSE*, 2015, pp. 403–414.

[18] J. Kerievsky, *Refactoring to Patterns*. Addison Wesley, 2004.

[19] G. Bavota, A. D. Lucia, M. D. Penta, and R. Oliveto, "An experimental investigation on the innate relationship between quality and refactoring," *Journal of Systems and Software*, vol. 107, pp. 1–14, 2015.

[20] T. Saika, E. Choi, N. Yoshida, S. Haruna, and K. Inoue, "Do developers focus on severe code smells?" in *Proc. of PPAP*, 2016, pp. 1–3.

[21] M. Lanza and R. Marinescu, *Object-Oriented Metrics in Practice*. Springer-Verlag New York, Inc., 2005.

[22] 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.

[23] Z. Xing and E. Stroulia, "Refactoring detection based on UMLDiff change-facts queries," in *Proc. of WCRE*, 2006, pp. 263–274.

[24] N. Tsantalis, V. Guana, E. Stroulia, and A. Hindle, "A multidimensional empirical study on refactoring activity," in *Proc. of CASCON*, 2013, pp. 132–146.