Does This Code Change Affect Program Behavior? –Identifying Nonbehavioral Changes with Bytecode–

Aoi Maejima*, Yoshiki Higo*, Junnosuke Matsumoto† and Shinji Kusumoto*
*Graduate School of Information Science and Technology, Osaka University, Japan
{a-maejim, higo, j-matumt, kusumoto}@ist.osaka-u.ac.jp

***Abstract*—Developers occasionally conduct some source code changes that do not affect program behavior. We call such changes *nonbehavioral changes*. In this research, we propose a technique for determining whether a given commit includes only nonbehavioral changes or not by checking the differences of bytecode on the commit. If the bytecode is not affected by the commit, the proposed technique determines that the commit includes only nonbehavioral changes. As a result of experiments on six Java open source projects, out of the commits in which Java source files were changed, the commits of 8.6%∼22.4% consisted of only nonbehavioral changes. We also found new 25 patterns of nonbehavioral changes compared to a previous study.**

***Index Terms*—MSR, change analysis, bytecode**

I. Introduction

There are many techniques to mine changes in code repositories. Mined changes have been used in various research studies such as defect prediction [1], [2] and code dependency detection [3], [4]. Some code changes such as modifying existing functions affect the program behavior. In contrast, other changes such as changing variable names do not affect the program behavior. In this research, we use term *behavioral changes* and *nonbehavioral changes* to show the former and the latter changes, respectively. Nonbehavioral changes are conducted to improve readability, maintainability, or other attributes of the source code.

In this paper, we propose a technique to classify code changes into behavioral/nonbehavioral ones by using bytecode. The key idea of our technique is that, if the bytecode after a given change is the same as the bytecode before the change, the change is regarded as a nonbehavioral one. Thus, our classification does not involve any subjectivity.

We conducted a small experiment on six open source projects to answer the following research questions.

RQ1 How do many nonbehavioral changes exist?
RQ2 What nonbehavioral changes were conducted?

TABLE I: Target projects and targer periods

Project	Start date	End date
kGenProg	09/Apr/2018	04/Nov/2019
Apache Ant	01/Jan/2017	22/May/2019
Spring Framework Core	15/Aug/2016	24/Jan/2020
Hibernate ORM Core	16/Dec/2016	31/Jan/2020
Apache Tomcat	04/Jul/2017	15/Jan/2020
Apache POI	26/Jan/2018	13/Jan/2020

II. Identifying Nonbehavioral Changes

In this research, we identify nonbehavioral changes according to the presence or absence of bytecode changes for given source code changes. We define that code changes that affect the byte are behavioral, and the others are nonbehavioral. Our technique creates another repository that is composed of Java source files and their decompiled bytecode. In the created repository, commits in which only Java source files are changed are nonbehavioral commits.

A. rjava Files

`rjava` files means files including decompiled bytecode. We use command '`javap -p -c [file]`' to decompile a bytecode file included in `class` files. Option '`-p`' specifies that all methods and fields, including private ones, are decompiled. Option '`-c`' specifies that all instructions in methods are decompiled, not only method signatures.

B. Creating a Census Repository

The following steps are iterated for all the commits included in a given repository to create a census repository including `java` and `rjava` files.

1) retrieve `java` files in a given commit.
2) compile `java` files to make `class` files.
3) decompile `class` files to make `rjava` files.
4) commit `java`/`rjava` files to the census repository.

C. Identifying Nonbehavioral Changes

We examine the change history of the census repository. If at least a `rjava` file is changed together with `java` files, the commit is regarded as a behavioral commit. Otherwise, the commit is regarded as a nonbehavioral one.

III. Experiment

We conducted an experiment on six Java projects. Table I shows the target projects and the target periods. The testing files in the projects are ignored in the experiment.

TABLE II: Number of commits in which `java` files with or without `rjava` files were changed

Project	# `.java` commits	# commits w/o `.rjava`
kGenProg	808	157 (19.4%)
Ant	323	64 (19.8%)
Spring	851	191 (22.4%)
Hibernate	1,253	114 (9.1%)
Tomcat	1,800	278 (15.4%)
POI	255	22 (8.6%)

978-1-7281-7303-0/20/$31.00 ©2020 IEEE
DOI 10.1109/COMPSAC48688.2020.0-119

A. Analysis for RQ1

The experimental results are shown in Table II. Our answer to RQ1 is that among the commits in which at least a `java` file was changed, the commits of 8.6%∼22.4% consist only of nonbehavioral changes.

B. Analysis for RQ2

We conducted visual confirmation for all the nonbehavioral commits for the projects. Table IV shows a list of the patterns of the nonbehavioral changes that we found. Table III shows the ratio of changes in the newly-detected patterns against all the nonbehavioral changes. The ratio is between 20.0%∼39.6%. Our answer to RQ2 is that we found 31 patterns of nonbehavioral changes, whereas the previous research [5] defined the six patterns. The upper six patterns in Table IV can be detected by the previous research too, while the remaining 25 patterns can be detected only by our technique.

TABLE III: Newly-detected nonbehavioral changes

Project	# all nonbehavioral changes	# newly-detected nonbehavioral changes
kGenProg	164	65 (39.6%)
Ant	75	15 (20.0%)
Spring	245	67 (27.3%)
Hibernate	139	37 (26.6%)
Tomcat	324	69 (21.3%)
POI	35	8 (22.9%)

IV. Conclusion

In this paper, we proposed a technique to identify nonbehavioral changes according to the presence or absence of bytecode changes. As a result of the investigation, we confirmed that in the six Java projects, the commits of 8.6%∼22.4% consisted of only nonbehavioral changes. Moreover, 25 new changes were defined as nonbehavioral changes compared to the previous research [5].

In the next step, we are going to apply our technique to reduce testing time in software developments with continuous integration.

References

[1] T. L. Graves, A. F. Karr, J. S. Marron, and H. Siy, "Predicting fault incidence using software change history," *IEEE Transactions on Software Engineering*, vol. 26, no. 7, pp. 653–661, July 2000.

[2] N. Nagappan and T. Ball, "Use of relative code churn measures to predict system defect density," in *Proc. International Conference on Software Engineering*, May. 2005, pp. 284–292.

[3] H. Gall, M. Jazayeri, and J. Krajewski, "Cvs release history data for detecting logical couplings," in *Proc. International Workshop on Principles of Software Evolution*, Sep. 2003, pp. 13–23.

[4] A. T. T. Ying, G. C. Murphy, R. Ng, and M. C. Chu-Carroll, "Predicting source code changes by mining change history," *IEEE Transactions on Software Engineering*, vol. 30, no. 9, pp. 574–586, Sep. 2004.

[5] D. Kawrykow and M. P. Robillard, "Non-essential changes in version histories," in *Proc. International Conference on Software Engineering*, May. 2011, pp. 351–360.

TABLE IV: Patterns of Nonbehavioral Changes

Pattern	Total	kGenProg	Ant	Spring	Hibernate	Tomcat	POI
Commits	457 (46.7%)	56 (34.1%)	42 (56.0%)	133 (55.1%)	61 (43.9%)	150 (46.3%)	15 (42.9%)
Formats	212 (21.7%)	26 (15.9%)	15 (20.0%)	30 (12.4%)	40 (28.8%)	93 (28.7%)	8 (22.9%)
Trivial Type Update	11 (1.1%)	2 (1.2%)	1 (1.3%)	4 (1.7%)		3 (0.9%)	1 (2.9%)
Local Variable Renames	22 (2.2%)	10 (6.1%)	1 (1.3%)	5 (2.1%)		5 (1.5%)	1 (2.9%)
Method Parameter Renames	12 (1.2%)	3 (1.8%)	1 (1.3%)	2 (0.8%)	1(0.7%)	4 (1.2%)	1 (2.9%)
`this.` Insertions/Deletions	7 (0.7%)	2 (1.2%)		4 (1.7%)			1 (2.9%)
`final` Insertions/Deletions	22 (2.2%)	19 (11.6%)		1 (0.4%)	1 (0.7%)	1 (0.3%)	
Import Statements	56 (5.7%)	18 (11.0%)	5 (6.7%)	14 (5.8%)	9 (6.5%)	8 (2.5%)	2 (5.7%)
Annotations	100 (10.2%)	25 (15.2%)	5 (6.7%)	31 (12.9%)	22 (15.8%)	16 (4.9%)	1 (2.9%)
`public` Insertions/Deletions for Methods in Interfaces	3 (0.3%)	1 (0.6%)			2 (1.4%)		
Unnecessary Casts	1 (0.1%)	1 (0.6%)					
Type in Lambda Expressions	1 (0.1%)			1 (0.4%)			
Empty Statements	5 (0.5%)	1 (0.6%)		1 (0.4%)	1 (0.7%)	1 (0.3%)	1 (2.9%)
De Morgan in Expressions	1 (0.1%)			1 (0.4%)			
Parentheses Expressions	11 (1.1%)			7 (2.9%)		3 (0.9%)	1 (2.9%)
Brackets for Blocks	11 (1.1%)				2 (1.4%)	8 (2.5%)	1 (2.9%)
If-Condition Consolidation	1 (0.1%)		1 (1.3%)				
`if-else-if` to two `if`	1 (0.1%)			1 (0.4%)			
Unnecesary `else`	1 (0.1%)			1 (0.4%)			
Generics	8 (0.8%)		3 (4.0%)	3 (1.2%)		2 (0.6%)	
Diamond Operator	4 (0.4%)		1 (1.3%)			3 (0.9%)	
Array Type Definition	1 (0.1%)			1 (0.4%)			
Reorder Modifiers	2 (0.2%)			1 (0.4%)		1 (0.3%)	
Empty String Concatination	1 (0.1%)					1 (0.3%)	
Init. w/ or w/o Static Block	1 (0.1%)					1 (0.3%)	
Iterator to Enhanced For-Block	1 (0.1%)					1 (0.3%)	
Autoboxing and Unboxing	19 (1.9%)					19 (5.9%)	
Number Type Specification	1 (0.1%)						1 (2.9%)
Use Constants	1 (0.1%)					1 (0.3%)	
`static` for Enum Declarations	3 (0.3%)					3 (0.9%)	
Semicolon for Enum Constants	1 (0.1%)						1 (2.9%)