Available online at www.ijpe-online.com

International Journal of Performability Engineering

vol. 16, no. 7, July 2020, pp. 1067-1077
DOI: 10.23940/ijpe.20.07.p9.10671077

An Empirical Study on the Impact of Code Contributor on Code Smell

Junpeng Jiang, Can Zhu, and Xiaofang Zhang*

School of Computer Science and Technology, Soochow University, Suzhou, 215006, China

Abstract

Code smells refer to poor designs that are considered to have negative impacts on the readability and maintainability during software evolution. Much research has been conducted to study the effects and correlations between them. However, software is a product of human intelligence, and the fundamental cause of code smell is developers. As a result, the research on the impact of code contributors on code smell appears vital in particular. In this paper, on 8 popular Java projects with 994 versions, we investigate the impact on code smells from the novel perspective of code contributors on five features. The empirical study indicated that the greater number of contributors involved, the more likely it is to introduce code smell. Having more mature contributors, who participate in more versions, can avoid the introduction of code smell. These findings are helpful for developers to optimize team structure and improve the quality of products.

Keywords: software quality; code smell; developer; software evolution

(Submitted on April 12, 2020; Revised on May 10, 2020; Accepted on June 8, 2020)

© 2020 Totem Publisher, Inc. All rights reserved.

1. Introduction

Code smell represents bad programming designs. The existence of code smell will adversely affect the understanding and maintenance of the program [1-3], and is considered to hinder the evolution and development of the software.

The existing research on code smell mainly focus on smell detection [4-6], smell evolution [7-9], and correlation between smell and file fault-proneness and change-proneness [10-12]. However, in the process of software evolution [7], it is the code contributors who operate the files or methods, that make the changes. As a result, human features are considered to have a great impact on software quality [13].

Tourani et al. [13] studied the impact of people's discussions about projects on defect propensity. Rahman et al. [14] explored the impact of code owner and developer experience on software quality. When it turns to the research of the impact of human features on code smell, Tufano et al. [15] studied the impact of developer status on smell introduction from three features: workload, whether the developer is the code owner, and whether the developer is a new participant. However, in this study, each code contributor is regarded as an independent individual, and the collaboration and teamwork of code contributors are not considered.

Therefore, in this paper, we conduct an extensive empirical study to investigate the impact on code smell from the perspective of code contributors. Specifically, we propose multiple code contributor features, i.e., the number of code contributors, closeness, maturity, experience and clustering. Then, we define the methods that potentially introduce code smell at the method level. Though a total of 994 release versions of 8 popular Java projects, we analyze the relationship between code contributors and code smells. Our empirical study results show that, at the file level, smelly files contain more code contributors than the non-smelly files; at the method level, the methods that potentially introduce code smell also have more contributors than other methods. In addition, the more contributors involved, the more likely it is to introduce code smell.

The more mature contributors, who participate in more versions in the project, can avoid the introduction of code smell.

* Corresponding author.
E-mail address: xfzhang@suda.edu.cn

The main contributions of this paper include: (1) Code contributors' features are defined in many ways, not only for individuals, but also for collaborations, which makes our research on code contributors more comprehensive and profound. (2) Impact analysis is done at multiple levels, including file level and method level. We analyze the influence of developers on smell from the surface to the inside, which deepens our understanding of code smell from the file level to the method level. (3) Experiments are studied on 994 versions of 8 projects, providing guidelines for building developer teams. Only through the analysis of experimental research on large data sets can our suggestions become more credible.

The rest of this paper is organized as follows: the approach and the study design are described in Section 2 and Section 3. Section 4 provides results and discussion of the study, as well as the threats and validity of our work. Then, some related work is introduced in Section 5. Finally, Section 6 presents the conclusion of our study results and discusses future work.

2. Approach

This section shows the approach used in our experiment, including the experiment process, collection of method change information, and the features used to describe code contributors.

2.1. Experiment Process

In order to understand the influence of code contributors on code smell, this paper adopts the following process, as shown in Figure 1.

- Git Blame command is used to obtain the features of code contributors, including the number, closeness, maturity, experience and clustering.
- DÉCOR [5] is used to detect the code smells in the file. With the information, the files containing the code smell are recorded as smelly files; the files without the code smell are recorded as non-smelly files.
- Abstract Syntax Tree (AST) is used to extract the methods in all files in each version. The code in each file is transferred to ASTs so that the methods that potentially introduce code smell can be defined by comparing the methods in adjacent versions.

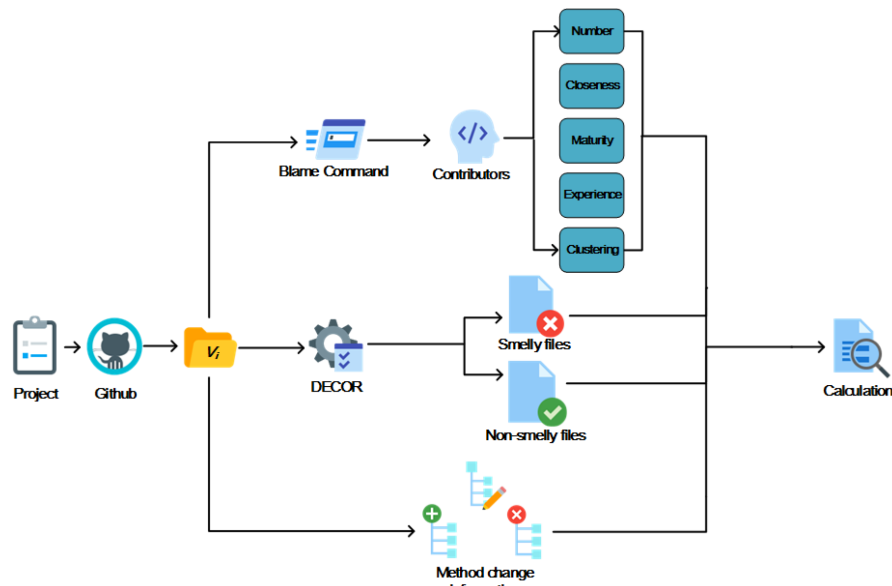

Figure 1. The process of this study

2.2. Collection of Method Change Information

The addition, modification or removal of certain methods can lead to the introduction or elimination of code smell. Therefore, we collect the different states of code smell in the process of software evolution, so as to judge whether the introduction or removal of code smells is a common phenomenon in the evolution of software. Table 1 shows the states of code smell, where S_i indicates the number of files with code smell, $Continus$ indicates the number of persistent smelly files, and $not\ Continuous$ indicates the number of not persistent smelly files.

Table 1. The state of code smell

Project name	$\|S_i\|$	Continuous	First_Last	First_Mid	Mid_Last	Mid_Mid	Not continuous
Framework	942	716	153	86	252	225	226
Security	328	274	0	43	85	146	54
Boot	605	407	4	17	143	243	198
Integration	604	477	0	21	170	286	127
Batch	398	191	3	10	90	88	207
Session	20	15	0	0	3	12	5
Data-mongodb	83	73	0	17	25	31	10
Data-gemfire	122	48	1	1	16	30	74

As shown in Table 1, code smells appear in various states in the process of software evolution, including the state of persistence and non-persistence in the version. In addition, this paper also details several states of persistent smelly files such as the file with code smells from the first version to the last version (First-Last), the file that code smells exist in the first version but disappear in the middle version (First-Mid), the file that the middle version has code smell and continues until the last version (Mid-Last) and the file where the appearance and disappearance of code smells occur in middle versions (Mid-Mid).

Changes in the appearance and elimination of code smell are often caused by changes in the methods. Therefore, by combining the occurrence and disappearance of code smell, as well as the information on the changes of methods, the methods that potentially introduce code smell are defined as potential smell-introducing methods (PSIMs in short).

First, the version information is established for the file with code smell. If a file j is smelly file, the version information of the smelly file is:

$$j: [(V_{a1}, V_{d1}), \cdots, (V_{an}, V_{dn})] \quad (1)$$

Where V_a represents the version in which the smell appears, V_d represents the version in which the smell is about to disappear, and (V_a, V_d) indicates the version interval where the code smell exists.

Then, PSIM is defined, as shown in Table 2. As for V_a, compared with the previous version, if a method in the file is modified between the two versions, or in the current version, the method is added as a new method. Then, define the method as PSIM. As for V_d, compared with the later version, if a method in the file is modified or removed between the two versions, then define this method as PSIM.

Table 2. Potential smell-introducing methods

Current version	Compare version	Method change information	PSIM
V_a	V_{a-1}	added-method	√
		modified-method	√
		-	-
V_d	V_{d+1}	modified-method	√
		removed-method	√
		-	-

2.3. Features of Mode Contributors

To explore the impact of code contributor on code smell, this section sets up the following five features of code contributors, as shown in Table 3.

Table 3. Features of code contributors

Features of code contributors	Description
Number of contributors	Reflects the number of contributors included in the method.
Closeness	Reflects the number of identical files that different contributors participate together, that is, whether the cooperation is close.
Maturity	Reflects how long contributors have been involved in the project.
Experience	Reflects the number of files that each contributor participate in the project.
Clustering	Reflects the degree of contributors clustering together in the project.

For each method in the current version i, this paper uses the analysis of social networks and quantifies the relevant features of the code contributors in the method as follow:

(1) Closeness: for all contributors of the method, establish an undirected weighted graphs with pairwise associations, where the weight represents the number of files that the two authors jointly participated in this version. The closeness of these two contributors is calculated as follows:

$$Closeness_{c1,c2} = \frac{contribution_{c1,c2}}{\sqrt{contribution_{c1} \times contribution_{c2}}} \tag{2}$$

Where $contribution_{c1,c2}$ indicates the number of files that $c1$ and $c2$ participate together, and $contribution_{c1}$ and $contribution_{c2}$ indicate the number of files that $c1$ and $c2$ separately involved in respectively. If the method contains only one contributor, the highest closeness value is 1.

(2) Maturity: the number of versions for the contributor $c1$ participating in the project between the first time version k and the current version i:

$$Maturity_{c1} = number\ of\ versions\ between\ i\ and\ k \tag{3}$$

(3) Experience: the number of files contributed by the contributor $c1$ in the current version:

$$Experience_{c1} = number\ of\ files\ c1\ contributed\ in\ version\ i \tag{4}$$

(4) Clustering: for the current version i, construct an undirectedconnection graph between code contributors, where if two contributors participate in at least one file, these two contributors are connected to each other. The clustering degree of code contributor $c1$ is calculated as follows:

$$Clustering_{c1} = \frac{2E(c1)}{n_{c1}(n_{c1} - 1)} \tag{5}$$

Where $E(c1)$ represents the number of edges connected to each other by the adjacent nodes of the contributor $c1$, and n_{c1} represents the number of adjacent nodes of $c1$.

3. Study Design

3.1. Research Question

In essence, the code contributor's operation has promoted software evolution, and it is also the root cause of the introduction of code smells. The features of code contributors provide developers a reliable indicator for making critical development decision in the software process.

Further, operations at file level and method level are both critical to software maintenance. Thus, analyzing the correlations between code contributors and code smells in different granularities is helpful for developers to make a proper plan of software refactoring.

In this study, we aim at answering the following research questions by analyzing the correlations between code contributors and code smells:

- RQ1: How does the number of code contributors change in smelly files and non-smelly files during the evolution of software projects?
- RQ2: Considering the introduction of smells at method level, which code contributor features are significantly related to the introduction of smells?
- RQ3: More specifically, what is the specific relationship between these features and the introduction of smells?

3.2. Subjects of Experiment

In the process of selection of experimental objects and data collection, we need to consider both data diversity and sufficiency.

Therefore, the selection of data set should according to the following principle: the experimental objects should have different scales and realize different functions to ensure diversity. In addition, to guarantee the data adequacy, there must be sufficient changed methods between two adjacent versions. Finally, 8 open-source Java projects in the Spring series are selected for empirical research. Their brief introductions are as follows:

- **Framework**: it provides a comprehensive programming and configuration model for contemporary java-based enterprise applications.
- **Security**: it is a powerful and highly customizable framework for authentication and access control.
- **Boot**: it is used to easily create stand-alone, production-level Spring-based applications, most of which require very little Spring configuration.
- **Integration**: it provides a simple model for building enterprise integration solutions.
- **Batch**: it provides a lightweight and comprehensive Batch processing framework for developing powerful Batch applications that are critical to the daily operation of enterprise systems.
- **Session**: it provides an API and implementation for managing user Session information.
- **Data mongodb**: it is part of the Spring Data project, which aims to provide a familiar and consistent Spring-based programming model for new Data storage.
- **Data gemfire**: it aims to make it easier to build highly extensible Spring-driven applications using Pivotal gemfire as the underlying distributed memory data management platform.

Table 4 summarizes the details of the experimental data set, showing their first and last versions, where $|v|$ represents the number of versions selected in the study. As shown in Table 4, we conduct our experiments on total 994 versions of 8 projects, which makes our results more convincing.

Table 4. Experiment subjects

Project name	First version	Last version	$\lvert v\rvert$
Spring framework	v3.0.0.M1	v5.1.5.RELEASE	146
Spring security	1.0.0	5.2.0.M1	123
Spring boot	v0.5.0.M1	v2.2.0.M1	123
Spring integration	v1.0.0.M1	v5.2.0.M1	178
Spring batch	spring-batch-1.0-m2	4.2.0.M1	91
Spring session	1.0.0.RC1	2.2.0.M1	53
Spring data mongodb	1.0.0.M1-MongoDB	2.2.0.M4	143
Spring data gemfire	v1.0.0.M1	2.2.0.M4	137

3.3. Analysis Methods

This paper adopts the Logistic Regression method to analyze the correlation between code contributors and code smell from multiple features.

Logistic regression is a generalized linear model. The dependent variable of logistic regression can be two-category or multi-category, while the two-category is more commonly used and easier to be explained. Multi-category can be processed by softmax. In this paper, the dependent variable has two values 0 and 1, which indicates whether it is a potential introduction of code smell. The formula of logistic regression is:

$$\pi(x_1, x_2, \cdots, x_n) = \frac{e^{w_0+w_1x_1+w_2x_2+\cdots+w_nx_n}}{1+e^{w_0+w_1x_1+w_2x_2+\cdots+w_nx_n}} \tag{6}$$

In Equation (6), $x_1, x_2, \cdots, x_n$ are independent variables in this paper, representing the number of code contributors, closeness, maturity, experience, and clustering in the method. $w_1, w_2, \cdots, w_n$ represent the weights of independent variables obtained by regression.

4. Experimental Results and Analysis

This section will show the result of our experiment of the 8 projects in detail and analyze the corresponding results to obtain relevant features that may lead to the introduction of code smell.

4.1. The Evolution of the Number of Code Contributors at File Level (RQ1)

This paper firstly presents the number of code contributors for smelly and non-smelly files at the file level to get a preliminary

understanding of the code contributor characteristics. Figure 2 shows the average number of code contributors included in two types of files. The x-axis in the figure represents the version, and the y-axis represents the average number of code contributors contained in the file, where the blue solid line and the green dotted line represent the average number of code contributors in smelly files and non-smelly files respectively. As shown in Figure 2, the smelly files contain more contributors than the non-smelly files in all projects. Besides, the average number of code contributors fluctuates with different amplitudes as the software evolves. However, the number of contributors in smelly files and the non-smelly files fluctuates consistently.

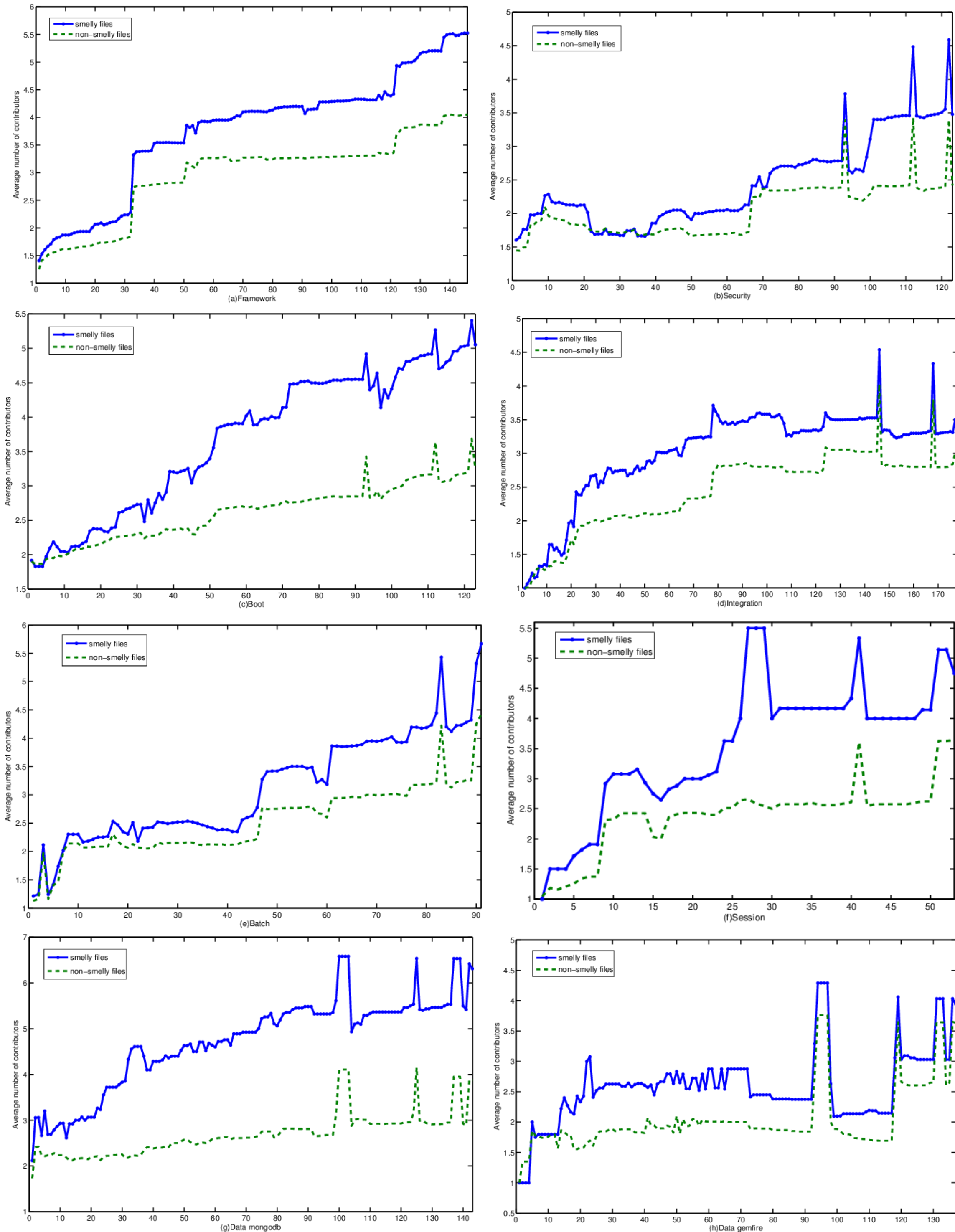

Figure 2. Average number of code contributors in smelly files and non-smelly files

It's not hard to notice that there are several versions in most of the projects where the number of contributors increased

sharply. By further analyzing these versions, it is found that when the number of files in these versions was relatively stable, the average number of files participated by new code contributors increased significantly, compared to other versions of the same project.

In summary, at the file level, smelly files contain a higher number of contributors than non-smelly files. However, it is difficult to make sure whether the number of code contributors will affect the introduction of code smells at the file level. Therefore, it is necessary to further conduct fine-grained research.

4.2. Multi-Feature Exploration of Code Contributors at Method Level (RQ2)

In order to further explain why smells are introduced, we investigate various features of the code contributors, including the number of contributors, closeness, maturity, experience and clustering. Then, this study uses logistic regression to show how these features are related to PSIMs. Table 5 shows the results of the analysis on code contributors based on multiple features. When the regression coefficient is positive, it means that the feature is positively correlated with PSIMs, and if it is negative, it means that the feature is negatively correlated with PSIMs. If the significance value is less than 0.05, it indicates that the feature has a significant impact on PSIMs.

Table 5. Analysis results of code contributors based on multiple features

		Framework	Security	Boot	Integration	Batch	Session	Data mongodb	Data gemfire
Number	regression coefficient	0.195	0.328	0.338	0.391	0.270	0.013	-0.031	0.350
	significance	0.000	0.109	0.000	0.000	0.008	0.970	0.838	0.018
Closeness	regression coefficient	-0.278	-1.051	0.051	0.341	-0.471	0.657	-1.449	-1.638
	significance	0.013	0.011	0.760	0.105	0.121	0.549	0.002	0
Maturity	regression coefficient	-0.004	-0.001	-0.005	-0.008	-0.026	-0.041	-0.018	-0.006
	significance	0.000	0.780	0.002	0.000	0.000	0.055	0.000	0.021
Experience	regression coefficient	0.000	0.000	0.000	0.000	0.001	-0.034	0.011	0.007
	significance	0.000	0.195	0.203	0.274	0.002	0.172	0.000	0.000
Clustering	regression coefficient	0.614	1.665	-0.275	-0.643	0.392	0.674	1.309	-1.386
	significance	0.005	0.004	0.215	0.000	0.061	0.621	0.031	0.014

From the results, it can be seen that in all projects except Data mongodb, the number of code contributors have a positive correlation with PSIMs, and it is statistically significant in the regression analysis model in 5 projects. Maturity presents a negative correlation with PSIMs in all projects. That means, the more versions the contributors are involved in, the more likely they are to avoid the introduction of smells, which is in line with our expectations.

The relationship between PSIMs and other features (closeness, experience and clustering) is inconsistent in different projects. Besides, it is not difficult to note that for Session, all the research features are not significant in the logistic regression model. We guess that it is because Session has a smaller number of versions as well as smaller version size than other projects. For Data mongodb, there are also the same problems on the fewer smelly files, and the impact of the number of contributors on the introduction of smell is inconsistent with other projects.

4.3. Single Feature Analysis of Code Contributors at Method Level (RQ3)

Based on the results of RQ2, the introduction of code smells is more related to the number and maturity of contributors. Thus, in this section, these two features are further analyzed separately. It mainly shows the concrete and intuitive connection between the two features of code contributors (the number and the maturity) and the introduction of smells by comparing the average values of these features in PSIMs with other methods. The study considered only files containing both PSIMs and other methods in the smelly files to ensure the accuracy of the data.

Figure 3 shows the average number of contributors in PISMs and other methods. In the figure, the x-axis, such as Framework-S, represents the PSIMs in Framework project. Framework-N represents other methods in Framework project. The y-axis represents the average number of code contributors. It is not difficult to see from Figure 3 that in all projects, the PSIMs contain a large number of code contributors, and the median value is also significantly higher than other methods.

Figure 4 shows the average maturity of contributors in PSIMs and other methods for each smelly file. The x-axis in Figure 4 is similar to Figure 3, and the y-axis represents the average maturity of the contributors. From the data distribution and the median value in Figure 4, it is shown that in all projects, the PSIMs have a lower average code contributor maturity. It is consistent with the conclusion that the contributor maturity of all projects is negatively correlated with PSIM in logistic regression.

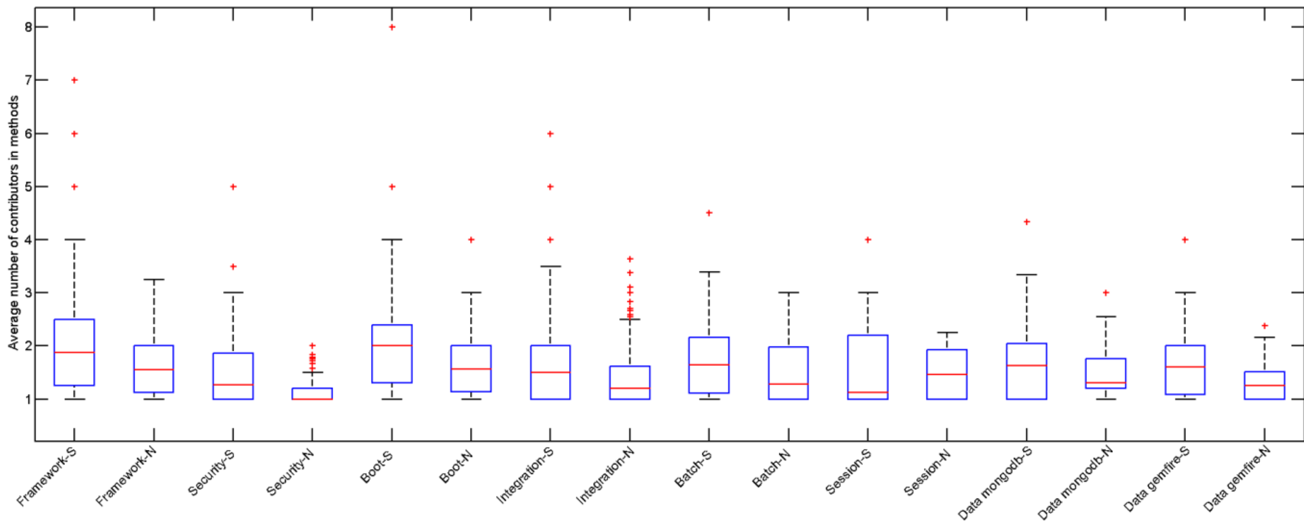

Figure 3. Average number of code contributors in PSIMs and other methods

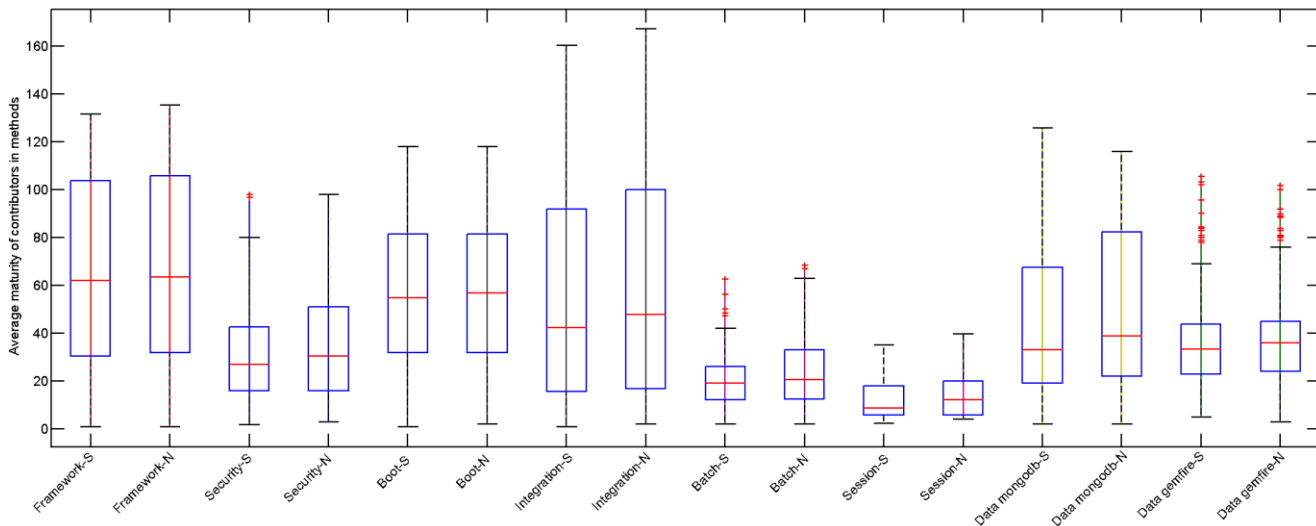

Figure 4. Average maturity of code contributors in PSIMs and other methods

To sum up, by investigating both at the file level and method level, we can turn to the conclusions that the number of code contributors who participate in the project can affect the quality of the code. The higher the number of developers involved, the more likely it is to lead to the irregular structure of code, so as to increase the maintenance cost of the code. Furthermore, both in multiple and single features analysis, code contributor maturity is negatively correlated with PSIMs. Thus, in the process of software development and evolution, developers who are familiar with the project itself can contribute to better programming.

4.4. Threats to Validity

Similar to many empirical studies, our work has some limitations in internal and external validity.

1) Internal Validity: one of the major threats to validity is the correctness of our experiment environment. Firstly, we chose DECOR to detect smells from recent tools [4, 6, 16-20]. The description of smells may have some subjective features,

which are the potential threats to our study validity. However, as a widely used smell detection tool, its recall reached 100%, and the average accuracy is more than 60%. Therefore, we think it is acceptable in our experiment that we used DECOR to detect code smells.

2) External Validity: we conducted our study on 8Java projects. These projects have a relatively large size and have been applied in different domains. We have explored a total of 994 versions. The data collected from these versions is large enough to support our experiment. We believe that due to enough experimental scale, our experimental results are reliable. Furthermore, our experiment was conducted at the file level and method level, which makes our experiment more acceptable and convincing.

5. Related Work

Code smells were first proposed by Fowler et al. [21]. They described 22 particular structures of code that contain design problems in software development and maintenance.

Code smell changes with the evolution of software. Chatzigeorgiou et al. [9] studied the evolution of smell in the system and found that in most cases, code smell persists in the development and evolution of software. Olbrich et al. [22] conducted a study on Gods and Shotgun Surgery, showing that the evolution of smell presents different characteristics in different stages and has a variety of change frequency and size. In our study, we not only focus on the smell changes with the evolution of software, but also compare the different code smells from the perspective of code contributors, so as to get the relationship between the kind of code smells and features of code contributors.

Khomh et al. [11] explored the impact of smells on change-proneness and fault-proneness in 54 versions of 4 projects. It was found that classes containing code smells tend to change more significantly in almost all versions of the study. The results also indicated that some smells had a high correlation with change-proneness and fault-proneness, such as AntiSingleton, while others had no significant correlation, and some correlation did not persist in the system, such as ClassDataShouldBePrivate. In this paper, inspired by their work, we define the potential smell-introducing method, making our research more convincing.

Tufano et al. [15] conducted a large-scale empirical study on the software evolution history of 200 open source projects to explore when and why smells were introduced. This study fills a gap in the academic understanding of code smells. This study figured out that most code smells are introduced when code instances are added to the system. By exploring the purpose of submission, the state of the project, and the state of the developer, they found that developers were more likely to introduce code smells when improving existing features or adding new features. In terms of the state of the project, most of the smells occurred in the last month before release, confirming that the deadline pressure on developers was one of the main causes of the smells. Our work was partially motivated by their conclusions. However, we aim to figure out whether the inside features of code contributors themselves are related to the code smells in their codes, instead of outside pressure. More specifically, we study which feature of code contributors will have a significant impact on the introduction of code smell.

6. Conclusion

There are few studies focused on the correlation between code smell and human features. In order to reveal the relationship behind them, we conducted an extensive empirical study to investigate the correlation between them.

In this paper, we define five features of code contributors both from individuals and cooperations, representing the features both from individuals and teams. The impact analysis is performed at multiple levels, including file level and method level, which provides a better understanding of the introduction and state of code smells. The empirical study is conducted on 994 versions of 8 projects, providing guidelines for building code contributor teams. Based on the results we obtained, we can obtain the following conclusions:

- At the file level, smelly files contain more contributors than non-smelly files.
- At the method level, the number and maturity of code contributors are significantly correlated to smell introducing, while other features present inconsistent characteristics in different projects.
- The greater number of contributors involved, the more likely it is to introduce code smell. Having more mature contributors who participate in more versions in the project can avoid the introduction of code smell.

The results and analysis provided by our empirical study suggest to streamline the structure of developers and choose high-maturity code contributors.

In future work, more empirical studies on other projects will be conducted to confirm our conclusions. Moreover, the specific human feature, such as centrality or ownership will be defined to discuss their impact on code smell.

Acknowledgements

This work was supported in part by the National Natural Science Foundation of China (61772263, 61772014), Suzhou Technology Development Plan (Key Industry Technology Innovation-Prospective Application Research Project SYG201807) and the Priority Academic Program Development of Jiangsu Higher Education Institutions.

References

1. S. M. Olbrich, D. S. Cruzes, and D. I. K. Sjøberg. "Are All Code Smells Harmful? A Study of God Classes and Brain Classes in the Evolution of Three Open Source Systems," in *Proceedings of 2010 IEEE International Conference on Software Maintenance*, IEEE, 2010
2. A. Yamashita and L. Moonen, "Exploring the Impact of Inter-Smell Relations on Software Maintainability: An Empirical Study," in *Proceedings of 2013 35th International Conference on Software Engineering* (*ICSE*), IEEE, 2013
3. F. Palomba, G. Bavota, M. D. Penta, F. Fasano, R. Oliveto, and A. DeLucia, "On the Diffuseness and the Impact on Maintainability of Code Smells: A Large Scale Empirical Investigation," *Empirical Software Engineering*, Vol. 23, pp. 1188-1221, 2018
4. J. Schumacher, N. Zazworka, F. Shull, C. Seaman, and M. Shaw, "Building Empirical Support for Automated Code Smell Detection," in *Proceedings of the 2010 ACM-IEEE International Symposium on Empirical Software Engineering and Measurement*, 2010
5. N. Moha, Y. G. Gueheneuc, L. Duchien, and A. F. Le Meur, "Decor: A Method for the Specification and Detection of Code and Design Smells," *IEEE Transactions on Software Engineering*, Vol. 36, No. 1, pp. 20-36, 2009
6. F. Khomh, S. Vaucher, Y. G. Gueheneuc, and H. Sahraoui, "A Bayesian Approach for the Detection of Code and Design Smells," in *Proceedings of 2009 Ninth International Conference on Quality Software*, IEEE, 2009
7. L. Aversano, G. Canfora, L. Cerulo, C. D. Grosso, and M. D. Penta, "An Empirical Study on the Evolution of Design Patterns," in *Proceedings of the 6th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering*, 2007
8. X. Zhang and C. Zhu, "Empirical Study of Code Smell Impact on Software Evolution," *Journal of Software*, Vol. 30, No. 5, pp. 1422-1437, 2019
9. A. Chatzigeorgiou and A. Manakos, "Investigating the Evolution of Bad Smells in Object-Oriented Code," in *Proceedings of 2010 Seventh International Conference on the Quality of Information and Communications Technology*, IEEE, 2010
10. C. Zhu, X. F. Zhang, Y. Feng, and L. Chen, "An Empirical Study of the Impact of Code Smell on File Changes," in *Proceedings of 2018 IEEE International Conference on Software Quality*, *Reliability and Security* (*QRS*), IEEE, 2018
11. F. Khomh, M. D. Penta, Y. G. Gueheneuc, and G. Antoniol, "An Exploratory Study of the Impact of Antipatterns on Class Change-and Fault-Proneness," *Empirical Software Engineering*, Vol. 17, No. 3, pp. 243-275, 2012
12. X. F. Zhang, Y. D. Zhou, and C. Zhu, "An Empirical Study of the Impact of Bad Designs on Defect Proneness," in *Proceedings of 2017 International Conference on Software Analysis*, *Testing and Evolution* (*SATE*), IEEE, 2017
13. P. Thongtanunam, S. Mclntosh, A. E. Hassan, and H. Iida, "Revisiting Code Ownership and its Relationship with Software Quality in the Scope of Modern Code Review," in *Proceedings of the 38th International Conference on Software Engineering*, 2016
14. F. Rahman and P. Devanbu, "Ownership, Experience and Defects: A Fine-Grained Study of Authorship," in *Proceedings of the 33rd International Conference on Software Engineering*, 2011
15. M. Tufano, F. Palomba, G. Bavota, R. Oliveto, M. Di Penta, A. De Lucia, et al, "When and Why Your Code Starts to Smell Bad," in *Proceedings of 2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, Vol. 1, IEEE, 2015
16. R. Marinescu, "Detecting Design Flaws via Metrics in Object-Oriented Systems," in *Proceedings 39th International Conference and Exhibition on Technology of Object-Oriented Languages and Systems*, IEEE, 2001
17. A. M. Fard and A. Mesbah, "Jsnose: Detecting Javascript Code Smells," in *Proceedings of 2013 IEEE 13th International Working Conference on Source Code Analysis and Manipulation* (*SCAM*), IEEE, 2013
18. R. Marinescu, "Detection Strategies: Metrics-based Rules for Detecting Design Flaws," in *Proceedings of 20th IEEE International Conference on Software Maintenance*, IEEE, 2004
19. R. Oliveto, F. Khomh, G. Antoniol, and Y. G. Gueheneuc, "Numerical Signatures of Antipatterns: An Approach based on B-Splines," in *Proceedings of 2010 14th European Conference on Software Maintenance and Reengineering*, IEEE, 2010
20. R. Marinescu, "Measurement and Quality in Object-Oriented Design," in *Proceedings of 21st IEEE International Conference on Software Maintenance* (*ICSM'05*), IEEE, 2005
21. M. Fowler, "*Refactoring*: *Improving the Design of Existing Code*," Addison-Wesley Professional, 2018
22. S. Olbrich, D. S. Cruzes, V. Basili, and N. Zazworka, "The Evolution and Impact of Code Smells: A Case Study of Two Open Source Systems," in *Proceedings of 2009 3rd International Symposium on Empirical Software Engineering and Measurement*, IEEE, 2009

Junpeng Jiang is an undergraduate student at the School of Computer Science and Technology at Soochow University, Suzhou, China. His research interests include software quality assurance and software testing.

Can Zhu is a Master's student at the School of Computer Science and Technology at Soochow University, Suzhou, China. Her research interests include software quality assurance and software testing.

Xiaofang Zhang is an Associate Professor at the School of Computer Science and Technology at Soochow University, Suzhou, China. Her research interests include intelligent software engineering and software testing.