Using Developers' Feedback to Improve Code Smell Detection

Mario Hozano
Federal University of Campina Grande
Dept. of Computing Systems
Campina Grande, Paraiba
mario@copin.ufcg.edu.br

Henrique Ferreira and Italo Silva
Federal University of Alagoas
Campus Arapiraca
Arapiraca, Alagoas, Brazil
italocarlo@nti.ufal.br

Baldoino Fonseca and Evandro Costa
Federal University of Alagoas
Computing Institute
Maceio, Alagoas, Brazil
baldoino@ic.ufal.br

ABSTRACT

Several studies are focused on the study of code smells and many detection techniques have been proposed. In this scenario, the use of rules involving software-metrics has been widely used in refactoring tools as a mechanism to detect code smells automatically. However, actual approaches present two unsatisfactory aspects: they present a low agreement in its results and, they do not consider the developers' feedback. In this way, these approaches detect smells that are not relevant to the developers. In order to solve the above mentioned unsatisfactory aspects in the state-of the-art of *code smells* detection, we propose the *Smell Platform* able to recognize *code smells* more relevant to developers by using its feedback. In this paper we present how such platform is able to detect four well known code smells. Finally, we evaluate the Smell Platform comparing its results with traditional detection techniques.

Categories and Subject Descriptors

D.2.8 [**Software Engineering**]: Metrics—*Product metrics*; K.6.3 [**Management of Computing and Information Systems**]: Software ManagementSoftware maintenance

Keywords

Refactoring, Code Smell Detection, Developer's Feedback

1. INTRODUCTION

There has been much research focusing on the study of *code smells*. Although these *smells* are sometimes unavoidable, in most cases, development teams should try to prevent and remove them from their code base as early as possible. Hence, many detection techniques have been proposed. [6] use predefined rules composed by software-metrics to identify *code smells*. Further, rules to detect *code smells* are extracted from software change history information [10], *Genetic Programming* [9], *Bayesian Belief Networks* [1].

SAC'15 April 13-17, 2015, Salamanca, Spain.

http://dx.doi.org/10.1145/2695664.2696059

However, two aspects, common to all detection approaches, are unsatisfactory because (i) the approaches present a low agreement in its results. Consequently, developers aiming to recognize software code parts that should be improved need to exploit the output of more than one tool; and, (ii) the approaches recognize *smells* without considering the developers' feedback. In this way, the approaches can indicate *smells* that are not relevant to the developers.

In order to solve the above mentioned unsatisfactory aspects in the state-of the-art of *code smells* detection, we propose the *Smell Platform* able to recognize *code smells* more relevant to the developers by using existing *smells* detection techniques and adapting them in order to consider the developers' feedback. In such context, we have applied our platform in releases of two *Java* open source projects, namely *GanttProject*[1] and *Xerces*[2]. Our aim was to analyze such projects in order to detect the *code smells* existing on them. In our experiment, our platform was able to detect *code smells* more relevant to the developers when compared with traditional detection techniques ones. In summary, the main contributions of this article are: (i) a platform to detect *code smell* more relevant to the developers (Section 3); and, (ii) An evaluation of the platform regarding *code smells* detection (Section 4).

2. RELATED WORK

Efforts aiming automatic techniques to detect *code smells* were initiated many years ago, the use of these techniques have helped developers to avoid a time expensive, unrepeatable and non-scalable detection process when performing in a manual way.

[6] and [3] propose rules based on software-metrics to detect *code smells*, such as: *God Class*, *Data Class* and *Feature Envy*. Such rules have been incorporated into *smell* identification tools (*inFusion*, *iPlasma*, *PMD*) widely used by developers. Nevertheless, these rules are defined in a general way and must be configured by developers in order to produce relevant results. However, according to [4], this configuration is a difficult, time-consuming and error-prone manual process.

To deal with the general rules limitations, some works have proposed alternatives to become the *smell* detection

[1]A free project scheduling and management app for Windows, OSX and Linux - http://www.ganttproject.biz/
[2]A library for parsing, validating and manipulating XML documents - http://xerces.apache.org/

more effective and precise. For instance, [8] and [7] proposes the use of relative thresholds in order to become the *smell* identification more project-sensitive. [10] proposes an approach to detect *code smells* by exploiting change history information mined from versioning system. In addition, recently, the use of intelligent techniques have been proposed by [1, 9, 5] to deal with uncertainty in *smell* detection. Despite these approaches have contributed to automate the *code smell* detection, they do not consider the developers' feedback while performing the identification process.

In [5], the developers' feedback is used to train a machine learning technique that is used to recognize *smells*. However, such feedback are not used iteratively, during the detection process. In [2] the detection process includes a training phase where developer can approve or reject the *code smells* detected by a classifier. A machine learning mechanism uses it to build a decision tree to detect *smells*. This work does not compare such approach with traditional detection techniques.

3. SMELL PLATFORM

The *Smell Platform* is able to detect *code smells* by using rule-based detection techniques and developers' feedback. Such feedback is obtained by analysing which detected *smell* the developer approves or rejects. These analysis are used to provide personalized rules, able to detect *code smells* more precisely for a specific developer. The platform's flow is illustrated in Figure 3.

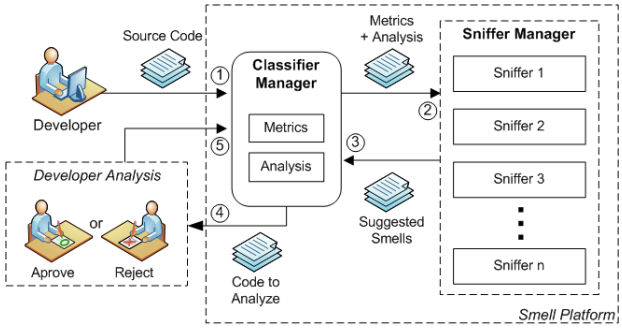

Figure 1: Smell Platform Execution Flow

The *Smell Platform* takes as input the source code of the project that will be analyzed (step 1). For each class/method of the analyzed project, the platform calculates object-oriented metrics that will be used by *Sniffers* (step 2). A sniffer uses the calculated metrics and an intelligent technique to recognize *code smells* existing in the analysed project. In addition, a sniffer is able to update its detection mechanism as soon as developers' feedback is obtained. After, in step 3, *Sniffers* detect the *code smells*, the *Classifier Manager* receives the detected *code smells*, filters them by removing the duplicated ones before showing them to developer. In step 4, the identified *smells* are analysed by developer. In this phase, the developer can approve or reject each recognized *smell*, individually. Finally, in step 5, all analysis are stored by *Classifier Manager* and they will help *Sniffers* to infer developer's understanding about *code smells*. These analysis will support *Sniffers* to create personalized rules and improve detection by starting a new cycle from step 2, automatically.

An implementation of the *Smell Platform* was built with four *Sniffers*. Each sniffer encapsulates a technique to detect a kind of *smell*. In this paper the *Sniffers* encapsulate a rule-based technique to detect *smells*. The initial rules to detect *Long Parameter List* and *Long Method code smells* were generated from the tools *PMD* and *Checkstyle*. The rules used to detect *God Class* and *Feature Envy* were based on [3], and are used by the tools *inFusion*, *iPlasma* and *PMD*.

The *Sniffer* is triggered to personalize the rule when a developer rejects a *smell* recognized by the platform. This new rule must be tuned to reject the analyzed *smell*. After the rule has been tuned, it is tested with all stored analysis and its efficiency is compared with older rules. The rule with best precision is used for the next detection process. In this case the precision is calculated as follows:

$$Accuracy = \frac{|TruePositives| + |TrueNegatives|}{|AnalyzedSmells|}$$

In this case, *True Positives* and *True Negatives* are correct classifications, where the platform and the developer agree whether a suggested *smell* is and not is a *smell*, respectively.

4. EXPERIMENT SETTINGS

The goal of this study is to assess whether the *Smells Platform* is an appropriate tool to recognize *code smells*. Therefore, the main research questions we aim to answer can be defined as follows: **RQ1**: Can personalized rules be inferred from developer's feedback?; **RQ2**: Can personalized rules improve *smell* detection?; **RQ3**: Can personalized rules, inferred from a project, be used to detect *smells* in other project?

4.1 Results and Discussion

Table 1 reports the accuracy resulting from the application of the *Initial Rule* (rules used by tools like *PMD*, *inFusion*, *Checkstyle*) and *Personalized Rule* (rules created by *Smell Platform*) to detect the *code smells Long Parameter List* existing in the *GanntProject* and *Xerces*. The first column identifies the developers involved in the experiment. The next two columns detail the accuracy of the *smells* identified in the *GanttProject* by using the *Personalized* and *Initial* Rules, respectively, followed by the difference between the accuracy of them. The last three columns detail the same results for *smells* detected in Xerces. Tables 2, 3, and 4 follow the same structure describing results for Long Method, God Class and Feature Envy detection, respectively.

Table 1: Long Parameter List Results

	GanttProject			Xerces		
Dev	PR	IR	PR - IR	PR	IR	PR - IR
1	-	1.00	-	-	0.95	-
2	-	1.00	-	-	0.85	-
3	1.00	0.80	0.20	1.00	0.15	0.85
4	1.00	0.90	0.10	0.20	0.15	0.05
5	1.00	0.95	0.05	0.65	0.55	0.10
6	1.00	0.95	0.05	0.20	0.15	0.05

Our platform was able to create 20 *Personalized Rules* of the 24 *smell* detection cases[3]. Therefore, our platform

[3]The complete results about this study are available at http://ic.ufal.br/sapiens

Table 2: Long Method Results

	GanttProject			Xerces		
Dev	PR	IR	PR - IR	PR	IR	PR - IR
1	1.00	0.75	0.25	0.65	0.65	0.00
2	0.95	0.90	0.05	0.60	0.60	0.00
3	0.90	0.70	0.20	0.85	0.75	0.10
4	0.85	0.80	0.05	0.85	0.85	0.00
5	-	0.95	-	-	0.75	-
6	1.00	0.85	0.15	0.95	0.70	0.25

Table 3: God Class Results

	GanttProject			Xerces		
Dev	PR	IR	PR - IR	PR	IR	PR - IR
1	-	0.90	-	-	0.20	-
2	0.85	0.75	0.10	0.55	0.40	0.15
3	0.80	0.70	0.10	0.25	0.25	0.00
4	0.85	0.60	0.25	0.70	0.55	0.15
5	0.85	0.80	0.05	0.45	0.45	0.00
6	0.75	0.55	0.20	0.45	0.45	0.00

was able to generate personalized rules in most of the detection cases by considering only the developer's feedback. In no time the developers have indicated what metrics and thresholds satisfy their understandings to detect *code smells*, answering **RQ1**.

To answer **RQ2** we need to verify if the *Personalized Rules* (PR) can detect *smells* more relevant to the developer than the *Initial Rules* (IR) used by existing detection tools. As discussed in previous section, our platform was able to create personalized rules in 83% of the detection cases. These personalized rules increased the accuracy of the *smells* detection in all cases where they were applied, as reported in the columns related to *GanttProject* results in Tables 1, 2, 3, and 4. In 6 cases involving *Long Parameter List* and *Long Method* detection, the Personalized Rules increase accuracy up to 100%.

Finally, to answer **RQ3**, we use the *Personalized Rules* (PR) generated from the analysis in the *GanttProject* to detect *smells* in the *Xerces*. We notice that even using the *Personalized Rules* generated from developer's feedback on *GanttProject*, the accuracy is improved when compared with detection using the Initial Rules (IR). Indeed, these *Personalized Rules* increased the accuracy of the *smells* detection in 12 of 20 cases, as reported in the last three columns in Tables 1, 2, 3, and 4. The better result was identified for Developer 3 in *Long Parameter List* detection, where the Personalized Rule increase the accuracy in 85% in the *smell* detection.

5. CONCLUSION

We have evaluated our platform over two open-source projects and we notice that it increased the accuracy of the *smells* detection up to 83% of the cases. In addition, even obtained from feedback over a specific project, the Personalized Rules increase the accuracy when used to detect smells in other project.

As future work we will apply other experiments to investigate how the platform can be used by varying developers, projects, and *code smells*. We are also extending the platform in order to use other information about the software such as change-history and bug report.

Table 4: Feature Envy Results

	GanttProject			Xerces		
Dev	PR	IR	PR - IR	PR	IR	PR - IR
1	0.50	0.45	0.05	0.70	0.50	0.20
2	0.65	0.60	0.05	0.80	0.80	0.00
3	0.70	0.65	0.05	0.35	0.35	0.00
4	0.95	0.85	0.10	1.00	0.75	0.25
5	0.70	0.60	0.10	0.50	0.35	0.15
6	0.55	0.20	0.35	0.65	0.50	0.15

6. REFERENCES

[1] F. Khomh, S. Vaucher, Y. G. Guéhéneuc, and H. Sahraoui. A bayesian approach for the detection of code and design smells. In *Quality Software, 2009. QSIC'09. 9th International Conference on*, pages 305–314. IEEE, 2009.

[2] J. Kreimer. Adaptive Detection of Design Flaws. *Electronic Notes in Theoretical Computer Science*, 141(4):117–136, Dec. 2005.

[3] M. Lanza, R. Marinescu, and S. Ducasse. *Object-Oriented Metrics in Practice*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2005.

[4] H. Liu, L. Yang, Z. Niu, Z. Ma, and W. Shao. Facilitating software refactoring with appropriate resolution order of bad smells. In *Proceedings of the the 7th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on The Foundations of Software Engineering*, ESEC/FSE '09, pages 265–268, New York, NY, USA, 2009. ACM.

[5] A. Maiga, N. Ali, N. Bhattacharya, A. Sabane, Y.-G. Gueheneuc, and E. Aimeur. SMURF: A SVM-based Incremental Anti-pattern Detection Approach. *2012 19th Working Conference on Reverse Engineering*, pages 466–475, Oct. 2012.

[6] R. Marinescu. Detecting Design Flaws via Metrics in Object-Oriented Systems A Metrics-Based Approach for Problem Detection. *International Conference and Technology of Object-Oriented Languages and Systems (TOOLS)*, pages 173–182, 2001.

[7] R. Marinescu. Detection Strategies: Metrics-Based Rules for Detecting Design Flaws. In *Proceedings of the 20th IEEE International Conference on Software Maintenance*, ICSM '04, pages 350–359, Washington, DC, USA, 2004. IEEE Computer Society.

[8] P. Oliveira, M. Valente, and F. Paim Lima. Extracting relative thresholds for source code metrics. In *Software Maintenance, Reengineering and Reverse Engineering (CSMR-WCRE), 2014 Software Evolution Week - IEEE Conference on*, pages 254–263, Feb 2014.

[9] A. Ouni, M. Kessentini, H. Sahraoui, and M. Boukadoum. Maintainability defects detection and correction: a multi-objective approach. *Automated Software Engineering*, 20(1):47–79, 2012.

[10] F. Palomba, G. Bavota, M. Di Penta, R. Oliveto, A. De Lucia, and D. Poshyvanyk. Detecting bad smells in source code using change history information. In *2013 28th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 268–278. Ieee, Nov. 2013.