cASpER: A Plug-in for Automated Code Smell Detection and Refactoring

Manuel De Stefano
m.destefano36@studenti.unisa.it
University of Salerno, Italy

Michele Simone Gambardella
m.gambardella24@studenti.unisa.it
University of Salerno, Italy

Fabiano Pecorelli
fpecorelli@unisa.it
University of Salerno, Italy

Fabio Palomba
fpalomba@unisa.it
University of Salerno, Italy

Andrea De Lucia
adelucia@unisa.it
University of Salerno, Italy

ABSTRACT

During software evolution, code is inevitably subject to continuous changes that are often performed by developers within short and strict deadlines. As a consequence, good design practices are often sacrificed, possibly leading to the introduction of sub-optimal design or implementation solutions, the so-called *code smells*. Several studies have shown that the presence of code smells makes the source code more change- and fault-prone, reduces productivity, and causes greater rework and more significant design efforts for developers. Refactoring is the practice that developers may use to remove code smells without changing the external behavior of the source code. However, it requires much time and effort and is poorly automated, often leading developers to prefer keeping low-quality code instead of spending time in designing and performing refactoring operations. To mitigate this problem and support developers throughout the process of code smell identification and refactoring, in this paper we present cASpER, a IntelliJ IDEA plugin that provides visual and semi-automatic support for detection and refactoring four different types of code smells.

Tool. Jetbrains: https://plugins.jetbrains.com/plugin/13659-casper
Video. https://youtu.be/HBWF8fFJM8s

KEYWORDS

Code smells, Refactoring, Automated Software Engineering.

ACM Reference Format:
Manuel De Stefano, Michele Simone Gambardella, Fabiano Pecorelli, Fabio Palomba, and Andrea De Lucia. 2020. cASpER: A Plug-in for Automated Code Smell Detection and Refactoring. In *AVI'20: ACM International Conference on Advanced Visual Interfaces, September 28–October 02, 2020, Island of Ischia, Italy*. ACM, New York, NY, USA, 3 pages. https://doi.org/10.1145/3399715.3399955

1 INTRODUCTION

Software life cycle inevitably demands continuous changes and enhancements [7], which too often require to be completed under strict deadlines. This too often leads developers to set aside old good design principles in favor of quick solutions, allowing the introduction of the so-called *code smells* [7], *i.e.*, sub-optimal design/implementation, which seriously impact on program comprehension, maintainability, as well as developer's productivity [7]. Refactoring represents the activity to remove code smells without altering the external behavior of the source code [7]. Unfortunately this activity is conducted either with a great manual effort, or with a limited automatic help, as few code smell detection and refactoring research proposals [1, 6] have become usable tools.

In this paper, we propose cASpER (Automated code Smell dEtection and Refactoring), a novel IntelliJ IDEA plugin that (1) integrates two state-of-the-art code smell detection approaches such as Decor [8] and Taco [9] to support the identification of four types of code smells (*i.e.*, Feature Envy, Misplaced Class, Blob and Promiscuous Package), (2) proposes refactoring recommendations implementing approaches previously proposed in literature [2, 3], (3) automatically modifies the source code according to the desired refactoring operations, (4) and visualize, in a single view, source code metrics, concepts and attributes. In the following sections, we briefly describe the features of the tool and a use case scenario.

2 CASPER'S FEATURES

In this section, we provide a brief description of cASpER features, focusing on the detection and the refactoring strategies adopted for each supported code smell (*i.e.*, Feature Envy, Misplaced Class, Blob and Promiscuous Package). The tool offers two kind of well-known and validated detection strategies: a structural one, relying on metrics computation (which are pointed out in the online appendix [10]), that is Decor [8], and a text-based one, Taco [9], which focuses on the textual content of the component under analysis.

Feature Envy and Misplaced Class represent a problem of a misplaced component, respectively at class (a misplaced method) and at package level (a misplaced class) [7]. Taco [9] detects them computing their conceptual similarity (textual cosine similarity) with external components and compares it with their container component. If the similarity with the most similar external container (envied container) is higher than the actual container, and the difference is higher than a given threshold, then the component is marked as smelly and a move method/class refactoring is suggested from the current container to the envied one. Decor [8], on the other hand, computes the component external references (method calls and dependencies respectively), and if they are more than the component internal one (class to methods of the same class or

AVI'20, September 28–October 02, 2020, Island of Ischia, Italy

ACM ISBN 978-1-4503-7535-1/20/09.
https://doi.org/10.1145/3399715.3399955

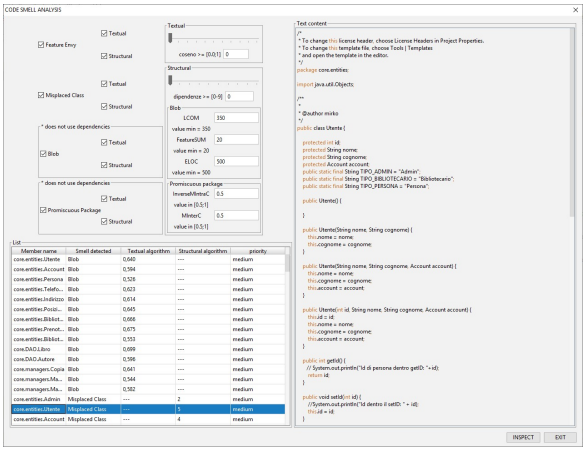

Figure 1: Results Wizard

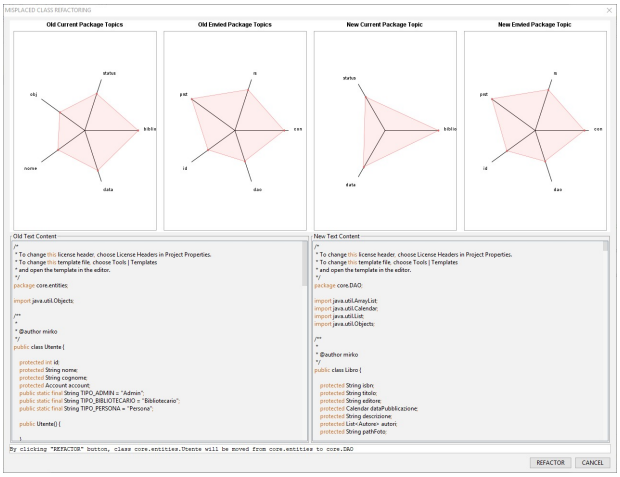

Figure 2: Refactoring wizard for Misplaced Class

dependencies to classes within the same package), than the component is marked as smelly. The refactoring algorithm is very simple for both: following Fowler's guidelines [7], and exploiting IntelliJ refactoring API, the smelly component is placed in the right spot.

The second couple of smell, Blob and Promiscuous Package, represent two example of a large, low cohesive, component, grouping together elements that are not related to each other, at class and package level respectively. Both the detectors use the concept of cohesion to find out if a component is smelly, computing it in different ways. Taco [9] computes the average textual similarity of all the internal components, while Decor [8], on the other hand, applies an heuristic based on the computation of structural metrics [4, 5], using LCOM, ELOC, WMC and NOA for classes, and MIntraC and MInterC for packages. If the cohesion is lower than a given threshold, the component is considered smelly and an extract class/package refactoring operation is suggested. The refactoring is based on the algorithms by Bavota *et al.* [2, 3]. The aforementioned code smell detection and refactoring approaches have been presented and validated in previous papers [2, 3, 8, 9], and so, cASpER accuracy is based on the accuracy of these approaches. Moreover it is important to remark that the automated refactoring is directly performed using the IntelliJ APIs: as such, the operations do not produce compilation errors.

3 CASPER AT WORK

For the sake of space limitation, we report only an example scenario of the tool usage in this section. In particular, we describe the whole procedure aimed to detect and correct occurrences of Misplaced Class on a system under development, Book-A-Book, a web application for libraries. Move Class Refactoring is supported by a three-step wizard.[1]

In the first step, the software engineer starts the code analysis selecting the *cASpER* command in the main menubar. Optionally, the engineer can configure the thresholds selecting the *configure* button in the same menu; otherwise, the default thresholds are used. Once the analysis is completed, a dialog box shows the results, as shown in Figure 1. On the left side of the window, there are as many checkboxes as the number of analyzed code smells, and some text boxes showing the thresholds used for the analysis. On the bottom side, there is a table showing the candidate smells found. The table is made up of 5 columns: the name of the candidate smelly component, the smell type, the degree of smelliness computed with both textual and structural algorithms, and the severity of the smell. Selecting one of the rows, the source code of the selected component is shown on the right side of the window. In the reported example, the engineer selects the class Libro, in the entities package, which the structural code smell detector, marks as *Misplaced*. In the second step, the tool shows a detailed view of the selected smelly class, the current package, and the envied package. In our scenario, Libro should be moved into the Dao envied package since this is the most conceptually similar class. If developer selects the *Find Solution* button to get the refactoring suggestion. cASpERshows a window with the recommended refactoring(Figure 2). Four radar maps show the five most frequent terms of the current package and the envied package, before and after the refactoring operation. Clicking on the *Refactor* button, the move class refactoring is performed, and the class is placed in the package suggested by the tool.

4 CONCLUSION

In this paper, we presented cASpER, open-source IntelliJ plugin for the automatic detection and refactoring of code smells. In our future works, we plan to employ the tool in a study aimed at assessing how developers react to cASpER recommendations. We also plan to add new code smell support as well as new detection strategies.

ACKNOWLEDGMENTS

Palomba gratefully acknowledge the support of the Swiss National Science Foundation through the SNF Project No. PP00P2_170529.

[1]The identification and refactoring of the other smells follow exactly the same wizard.

REFERENCES

[1] Jehad Al Dallal. 2015. Identifying refactoring opportunities in object-oriented code: A systematic literature review. *Information and software Technology* 58 (2015), 231–249.

[2] Gabriele Bavota, Andrea De Lucia, Andrian Marcus, and Rocco Oliveto. 2013. Using structural and semantic measures to improve software modularization. *Empirical Software Engineering* 18, 5 (2013), 901–932.

[3] Gabriele Bavota, Andrea De Lucia, Andrian Marcus, and Rocco Oliveto. 2014. Automating extract class refactoring: an improved method and its evaluation. *Empirical Software Engineering* 19, 6 (2014), 1617–1664.

[4] S. R. Chidamber and C. F. Kemerer. 1991. Towards a metrics suite for object oriented design. In *Proceedings of 6th ACM Conference on Object-Oriented Programming Systems Languages and Applications (OOPSLA)*. 197–211.

[5] S. R. Chidamber and C. F. Kemerer. 1994. A Metrics Suite for Object Oriented Design. *IEEE Transactions on Software Engineering* 20 (June 1994), 476–493.

[6] Eduardo Fernandes, Johnatan Oliveira, Gustavo Vale, Thanis Paiva, and Eduardo Figueiredo. 2016. A review-based comparative study of bad smell detection tools. In *Proceedings of the 20th International Conference on Evaluation and Assessment in Software Engineering*. 1–12.

[7] M. Fowler. 1999. *Refactoring: improving the design of existing code.* Addison-Wesley.

[8] Naouel Moha, Yann-GaÃńl GuÃĺhÃĺneuc, Laurence Duchien, and Anne-Francoise Le Meur. 2010. DECOR: A Method for the Specification and Detection of Code and Design Smells. *IEEE Transactions on Software Engineering* 36 (01 2010), 20–36. https://doi.org/10.1109/TSE.2009.50

[9] Fabio Palomba, Annibale Panichella, Andrea De Lucia, Rocco Oliveto, and Andy Zaidman. 2016. A Textual-based Technique for Smell Detection. https://doi.org/10.1109/ICPC.2016.7503704

[10] Manuel De Stefano, Michele Simone Gambardella, Fabiano Pecorelli, Fabio Palomba, and Andrea De Lucia. 2020. *cASpER: A plugin for Automated Code Smell Refactoring Detection And Refactoring - Online Appendix.* https://doi.org/10.6084/m9.figshare.12046377