International Journal of Advanced Science and Technology
Vol.60, (2013), pp.25-32
http://dx.doi.org/10.14257/ijast.2013.60.03

A Comparative Study on Code Smell Detection Tools

Almas Hamid, Muhammad Ilyas, Muhammad Hummayun and Asad Nawaz

Department of Computer Science and IT, University of Sargodha, Sargodha, Pakistan
Almas.hamid@yahoo.com, m.ilyas@uos.edu.pk, hummayunmalik101@yahoo.com, sanbhal88@gmail.com

Abstract

Refactoring is a technique to make a computer program more readable and maintainable. A bad smell is an indication of some setback in the code, which requires refactoring to deal with. Many tools are available for detection and removal of these code smells. These tools vary greatly in detection methodologies and acquire different competencies. In this work, we studied different code smell detection tools minutely and try to comprehend our analysis by forming a comparative of their features and working scenario. We also extracted some suggestions on the bases of variations found in the results of both detection tools.

Keywords: *Refactoring, Code Smell, Eclipse, JDeodrant, inCode*

1. Introduction

Refactoring has become a well known technique for the software engineering community. Martin Fowler has defined it as a process to improve the internal structure of a program without altering its external behavior [1]. Frequent refactoring of the code helps programmer to make the code more understandable, find bugs and make it suitable for the addition of new features and to program faster. Above all that, it improves the design of the software and therefore the overall quality of the software [1]. Refactoring can be done manually as well as automatically. Extensive literature is available on refactoring of the object oriented-programs and a number of tools are available for the automatic refactoring of the code.

Refactoring has a special relationship with the concepts of reverse engineering and agile software development. One of agile software development models, eXtreme Programming (XP), proposed by beck [3], considers refactoring as one of its essential features. Refactoring continuously improves the design of the software and helps the evolution and incremental development of the software.

Different steps for the activity of refactoring have been proposed by different researchers. Tom Mens *et al.*, [4] described six activities in the process of refactoring. These are: Identifying; where refactoring should be done, determining which refactoring(s) should be applied to the identified places, ensuring that the applied refactoring preserves behavior, applying the refactoring, assessing the effects of refactoring on the quality of the software and process, assessing the consistency between the code and other software artifacts

Bad smells are design flaws or structural problem of software that can be handled through refactoring. The term refactoring was first proposed by Kent Beck while helping martin Fowler [1]. Later Fowler did much work in this context and this work is still in progress.

ISSN: 2005-4238 IJAST

In [7] smell taxonomy is presented by Martin Fowler. He identified 22 bad smells ranging from ordinary bad smells like code duplication and Long parameter list to more complex smells like God Class and Feature Envy. He divided these 22 code smells into 7 categories based on their similarity. For example dispensable is one of those 7 categories that includes lazy class, data class, duplicated code dead code and speculative generality smells based on the similarity that all these incorporate redundant data. Other researchers also classified code smells in different ways. One way is to classify the code smells into two categories: smells within a class (*e.g.*, as long method, switch statement, comments and code duplication) and smells outside the class (*e.g.*, as data class and middle man *etc.*,) [6].

A variety of software tools have been developed for the automated detection of bad smells and they differ in their capabilities and approaches. Determining whether some piece of code contains bad smell(s) is somewhat subjective and still there is a lack of standards.

In this work, a comparative study is carried out regarding two bad smell detection tools namely JDeodorant and inCode. Their detection methodology is discussed in greater detail and variations in results are noted. We selected Feature Envy and God class code smells to do work with. Both tools are evaluated on these two smells.

2. Literature Review

Steve *et al.*, [8] described analysis of code smells. They used bad smell taxonomy described in [7, 9] and a bespoke software tool [10] to find number of refactoring required for each of 22 bad smells. According to his analysis, Bloaters is the smell category that needs most effort because it requires larger number of refactoring to remove from code. The Change Preventers needs least effort because it requires lowest average number of refactoring to eradicate from code. They claimed that bad smells are deceptive in nature mean that they hide the real effort that would be required to remove that smell when nested refactoring are involved. Bad smells that requires most effort to remove are best understood by developers and, conversely, bad smells requires less effort to remove are least understood by developers.

In [11] Foutse *et al.*, presented that classes with code smells are more change prone then the classes without code smells. They further showed that the correlation between particular type of code smell and change proneness. They used DECORE [12] approach to detect code smells and applied different mathematical techniques to conclude the results. Two systems from different domains were selected for experimentation. Both were open source systems namely Azureus and Eclipse [11]. From 13 releases of Eclipse and 9 releases of Azureus they showed that classes with bad smells are more change prone then the classes without bad smells. They further proved that a particular kind of bad smells lead to change-proneness. However this study does not concern the type and amount of change and it is limited to only two systems. If it is generalized to more systems it may give different outcomes.

Evolution of three bad smells on two open source system was studied in [19]. Consecutive version of jFlex and jFreechart were taken into account to perform the analysis. The bad smells namely Long method, Feature envy and State checking were extracted using JDeodorant. The result showed that the number of bad smells in system increase as the system evolves over the time; many of the bad smells remain in the code up to latest versions of the system; some of the bad smells that are removed were not a result of targeted refactoring application rather they are removed in an effort of adaptive maintenance.

Fontana *et al.*, [20] presented a report on on smell detection tools. They stated result of six bad smell detection tools; experimentation was performed on 5 versions of Gantt Project [21]. As none of the bad smell shared by all tools, that's why they perform their analysis on a subset of the tool. On the basis of their experiments and results they concluded that; main critical aspect is to have no access to detection rules and metrics thresholds. They proposed that bad smell detection tools should have the possibility to set and change the metrics threshold.

3. Experimental Work

We performed our analysis on GUI based multimedia application using two code smell detection tools. The scenario of experimentation is given in subsequent paragraphs.

3.1. Selection of Bad Smell Detection Tools

A variety of smell detection tools are available for code smells detection. Below is the description of those tools that we have used in our comparative study. These tools can applicable only for java code.

3.1.1. JDeodorant: JDeodorant [15, 16] is an eclipse plug in that is so far identifies four kinds of bad smells, namely "Feature Envy", "Type Checking", "Long Method" and "God class". This tool uses the ASTParser API of eclipse Java Development Tool to detect bad smell from the source code and ASTRewrite API to apply refactoring. In addition to detect smells and apply refactoring, JDeodorant encompasses many other features including transformation of expert knowledge to fully automated processes, pre evaluation of effect for each suggested solution, user guidance in comprehending the design problems and user friendliness [14]. Currently a new feature has been embedded in JDeodorant, which is execution of this tool in batch processing mode [13]. In this mode, no interaction with eclipse interface is required. Rather all open Java projects can be analyzed without user interaction [2, 5].

3.1.2. inCode: inCode is also an eclipse plug-in for smells detection. The main feature of this tool is to support programmers to program in code smells aware programming environment. It works in the background of eclipse. During programming, if programmer writes any bad structure, than it shows these smells as, "eclipse show error" and warnings in the shape of red color blocks along with code. inCode detects 4 bad smells *i.e*. Feature envy, God class, Duplicate code and Data class. These bad smell detections are based on object-oriented metrics. The greater part is that user doesn't have to interact with the underlying metrics directly. Metrics do their job under the hood, while you zip directly to the useful conclusions. Moreover, it also has functionalities of construction of Metrics Pyramid [17]. The pyramid shows the values of different metrics and their correspondence within the software.

3.2. Subject System

Xtreme Media Player [18] is a free cross-platform media player, licensed under GPLv2. It supports different audio formats and playlist formats. It also provide remote music playback via URL. It can read general tag info and specific tag info depending on the audio format. Attractive interface having different playing modes and

Multilanguage support is provided. Playlist manager and keyboard shortcuts are also provided. User can update the application automatically.

Its first version was 0.5.1 released on March 27, 2008 after that many versions have been released. Latest version is 0.7.0 released on September 12, 2011. The version we have selected for our case study is xtrememp-0.6.3 released on January, 31, 2009 and xtrememp-0.6.6 released on October 23, 2010 [18].

Metrics for both version of XtremeMP is given in Table 1. These metrics are calculated with a tool namely Metrics [23].

Table 1. Xtreme-MP Project Metrics

Metrics	XtremeMP-0.6.3	XremeMP-0.6.6
Total lines of code	6797	7956
Number of Packages	13	17
Number of Classes	63	71
Number of interfaces	4	8
Number of Methods	459	498
Method lines of code	4146	5142
Weighted methods per class	1142	1341
Number of static methods	60	70

3.3. Selection of Bad Smells

Many research groups are working on the development of automated refactoring tools. Currently, different refactoring tools are available which cover different aspect of bad smells. There are almost 93 refactoring available in the literature. Though, many tools provide some of these refactoring, none of the tool provides all these refactoring.

We selected two refactoring tools (details given in the last section). These tools cover multiple bad smells detection, but in the scope of our study, we have selected only bad smells named, “Feature Envy” and “God Class”. Short description of both bad smells is given in later paragraph.

3.3.1. Feature Envy: Feature Envy is a violation of the principle of object oriented about grouping behavior with related data. “It occurs when a method is more interested in a class other than the one it actually is in”. [1] Most important cures to eradicate bad smells are Move Method and Move Field refactoring. [1]

3.3.2. God Class: God Class is a violation of the principle of object oriented that a class should implement only one concept [22]. It is often trying to do much; it often shows up as to many instance variables [1]. Due to important role in the system, many other system classes can be dependent on it. As consequence changes in God Class during software maintenance and evolution can lead developers to problems. The refactoring that use to remove bad smell are Extract Class, Extract Subclass, Extract Interface and Duplicate Observed Data [1].

4. Results Findings

Experimentation was performed with default threshold values. Detection results are written in Table 2.

Table 2. # of Found Smells with Selected Tools in both versions of Xtreme MP

Tool	Feature Envy		God Class	
	Ver. 0.6.3	Ver. 0.6.6	Ver. 0.6.3	Ver. 0.6.6
JDeodorant	1	3	14	16
InCode	2	2	2	2

4.1. Feature Envy Discussion

On examining the results, we found that the instances of feature envy bad smells detected by both tools are entirely different. Feature Envy smells detected by InCode are static methods, they do not manipulate any data of their source class but they process data of other system classes. According to object oriented design heuristics and principles; method must be placed in the class, which data it manipulates more. That is the reason InCode detects these methods as bad smell.

The JDeodorant does not detect these static methods as bad/code smell. The JDeodorant detects three non static methods as feature envy bad smells. But these detected bad smells are not present in the results of InCode.

The non static methods detected as feature envy bad smell by JDeodorant; use attributes (data member) of source class but these attributes are of reference type. Type of these attribute is other system classes. So, these methods with the help of reference typed attributes manipulate data of other system classes.

That is the reason JDeodorant identifies these methods using its distance based approach. In which, it measure the distance between the method and the class, expressing the dissimilarity between the set of entities (method and attributes) accessed by method and set of entities belonging to class.

The InCode does not detect these methods as feature envy bad smell. Here a question arises from these results of InCode for feature envy; which are given below:

Does InCode detect non static methods as feature envy smell or not?

Current results do not give much information about this question. We make a separate version of this API by applying the God Class refactoring on the Xtreme-MP 0.6.6 with JDeodorant. By detecting Feature Envy bad smell from the refactored version of API with InCode, we get some results which somehow give some satisfactory results about this question. By these result we found that InCode detects non-static methods as feature Envy bad smell. Due to no access to InCode's documentation for its bad smell detection rules and metrics threshold; we are unable to understand the reason why InCode does not detect non-static methods detected by JDeodorant.

4.1.1. Concluding Remarks on Feature Envy's Results: As the Martin Fowler wrote in his book:"Method must be on that object which data it uses."

As we see this guideline, then we get thought of only non-static methods as feature envy; because only non-static methods are on the object. The static methods are not associated with any particular object, because they are shareable behaviors of all objects. To access them we do not need object of the class in which it resides. They are accessed with their class names in the system.

No doubt the guideline of Martin, has sense that consider only non-static methods as feature envy. But if we see the main philosophy behind this which is to prevent the ripple effects of changes in software system. To implement this philosophy in system design, we grouped methods to that class which data it most uses. Due to this reason, static and non–static methods are equally important.

So, as JDeodorant detects very efficiently non-static as feature envy. It must incorporate the non-static methods also in the category of feature envy. The InCode also needs improvements in its methodology to detect feature envy smell. In the case when the non-static methods use the reference type fields (attributes) to access the data of other system classes.

At last none of both has concrete process to detect feature envy bad smell. Both have deficiencies in their algorithms. The missing concepts of both are important; if they consider them they can improve the quality of software.

4.2. God Class Discussion

By evaluating the result of both tools we found large difference between their results. The InCode detects two and JDeodorant detect fourteen God Classes from the both versions of Xtreme-MP.

On further in depth analysis we found that both tools have entirely different approaches to identify God Classes.

The God Classes dentified by JDeodorant also has those two God Classes which identified by InCode. But they rank some other detected God Class smells more important for the design quality of the examined project with the help of Entity Placement Metrics [23].

The InCode use the metrics based approach to detect God class smells. It checks the encapsulation through cohesion and coupling metrics and the cumulative complexity of the methods of the class to identify the God Class smells.

The JDeodorant uses the agglomerative clustering technique. In which it extract information from the source code of the examined project to calculate the distance between the entities (methods and attributes) of the classes and then apply the clustering algorithm to identify the coherent group of entities as the opportunities as God Class [13].

A large difference exists in the scope and goal of the approaches used by both tools. InCode goal is to identify such system classes as God Class which has lack of cohesion and high complexity. To refactor the God Classes detected by InCode we need some further processing to determine where we can apply which refactoring. For this we need to determine; this God Class is either Data God Class or behavioral God Class. After classification we apply one of the refactoring from Move Field Refactoring, Move method refactoring, Extract class Refactoring.

But all these things described in above paragraph cannot be attained automatically from the current methodology used by InCode. For this we must need some manual interaction in detection and classification of God Class, when we using InCode to detect and refactor God Classes.

Goal of the methodology used by JDeodorant for God Class is to identify the opportunities of extract class refactoring. In the mechanism and way of representation of the result of the God class smell detection to software engineers with Entity Placement Metrics is best.

But the extract class refactoring is not the solution in all cases and conditions for the God Class.

While performing experimentation for finding Feature envy smell through InCode, we found an interesting result about God class. In this experiment we refactor single God Class smell (xtrememp.XtremeMP) with JDeodorant by extracting a class (Xtreme-MP. XtremeMPProduct).After this refactoring when we again detect God Class bad smell form this refactored version of API with both tools; they detect no god class. On

the basis of this finding one point is clearer that InCode detects only two classes as God Class but they are the ones that have great impact on quality of code.

4.2.1. Concluding Remarks on God Class's Results: Both tools do not provide comprehensive solution. There is a need of such approaches which identify the God Classes and then identify opportunities of all types of refactoring solutions that proposed by author of refactoring [1] for God Class in different cases of God Class.

5. Conclusion and Future Work

In this work we analyze two code smell detection tools and comparison is presented. We evaluated why tools depict variation in results and what specific parameters they used for detecting a particular smell.

We found that basic difference in results lies due to use of different approaches for smell detection. Both tools have their own efficiencies and deficiencies, which we endeavored to comprehend in this work. On the basis of our finding we say that though, there is lack of tool support in context of smell detection and refactoring, available tools are not mature enough. Besides development of new tools current tools also need to be revised.

In the future, we are interested in investigating the change proneness of code having God Class and Feature Envy bad smells and impact of this change proneness on maintainability of software. A prototype is in progress which will provide state-of-the-art solution of the problems identified in this study and in prior research.

References

[1] M. Fowler, K. Beck, J. Brant, W. Opdyke and D. Roberts, "Refactoring: Improving the Design of Existing Code", Addison-Wesley, **(1999)**.
[2] M. Munro, "Product Metrics for Automatic Identification of "Bad Smell" Design Problems in Java Source-Code", Software Metrics, 11th IEEE International Symposium, **(2005)**, pp. 15.
[3] K. Beck, "Extreme Programming Explained: Embrace Change", Addison-Wesley. ISBN 978-0-321-27865-4, **(1999)**.
[4] T. Mens and T. Tourw´e, "A survey of software refactoring", IEEE Transactions on software engineering, vol. 30, no. 2, **(2004)**, pp. 126-139.
[5] N. Roperia, "JSmell: A Bad Smell detection tool for Java systems", Long Beach, CA, USA: ProQuest Dissertation and Theses, **(2009)**.
[6] Coding Horror: Code Smells. **(2012)** http://www.codinghorror.com/blog/2006/05/code-smells.html [Accessed at:, **(2012)** March 4]
[7] M. Mantyla, J. Vanhanen and C. Lassenius, "A Taxonomy and Initial Empirical Study of Bad Smells in Code. **(2003)**", Proceedings of the International Conference on Software Maintenance, pp. 381.
[8] S. Counsell, H. Hamza and R. M. Hierons, "An Empirical Investigation of Code Smell 'Deception' and Research Contextualisation through Paul's Criteria", **(2010)** Journal of Computing and Information Technology-CIT 18, **(2010)**, vol. 4, March 4, pp. 333-340.
[9] M. Mantyla, "Bad Smells in Software — A Taxonomy and an Empirical Study", Master's Thesis, Helsinki University of Technology, Software Business and Engineering Institute, **(2003)**.
[10] H. Hamza, S. Counsell, G. Loizou and T. Hall, "Code Smell Eradication and Associated Refactoring", Proceedings of the European Computing Conference (ECC), Malta, **(2008)** September.
[11] F. Khomh, M. D. Penta and Y. Guéhéneuc, "An Exploratory Study of the Impact of Code Smells on Software Change proneness", Proceedings of the 16th Working Conference on Reverse Engineering (WCRE), Lille, France, IEEE Computer Society Press, **(2009)** October 13-16.
[12] N. Moha, Y. Guehenue, L. Duchien and F. L. Meur, "DECORE: A method for specification and detection of code and design smells", Software Engineering, IEEE Transactions, vol. 36, no. 1, **(2010)** January-February, pp. 20-36.
[13] Jdeodorant, http://jdeodorant.com/ [Accessed: April.17, 2012].

[14] Eclipse Marketplace," online Available: http://marketplace.eclipse.org/content/jdeodorant [Accessed: April 17, 2012].
[15] M. Fokaefs, N. Tsantalis and A. Chatzigeorgiou, "JDeodorant: Identification and Removal of Feature Envy Bad Smells", IEEE International Conference on Software Maintenance, **(2007)** October, pp. 519-520.
[16] T. Chaikalis, N. Tsantalis and A. Chatzigeorgiou, "JDeodorant: Identification and Removal of Type-Checking Bad Smells", 12th European Conference on Software Maintenance and Reengineering, **(2008)**, pp. 329-331.
[17] InCode website problem detection on fly [April.17, 2012].
[18] Sourceforge, http://xtrememp.sourceforge.net/ [Accessed: May.13, 2012].
[19] A. Chatzigeorgiou and A. Manakos, "Investigating the Evolution of Bad Smells in Object-Oriented Code", Quality of Information and Communications Technology (QUATIC), 2010 Seventh International Conference, pp. 106-115, **(2010)** September 29-October 2.
[20] F. Fontana, E. Mariani, A. Mornioli, R. Sormani and A. Tonello, "An Experience Report on Using Code Smells Detection Tools", ICSTW '11: Proceedings of the 2011 IEEE Fourth International Conference on Software Testing, Verification and Validation Workshops, **(2011)**.
[21] Gantt Project, Available: http://www.ganttproject.biz/ [Accessed: April 12, 2012].
[22] R. C. Martin, "Agile Software Development: Principles", Patterns and Practices Prentice Hall, Upper Saddle River, NJ, **(2003)**.
[23] Metrics 1.3.6", http://metrics.sourceforge.net/, [July 15, 201].

