A Survey on Code Smell Detection with a Perspective of Software Design

S.Nandhini[1] Ms.E.Baby Anitha[2] Ms.D.Sandhiya[3]
[1,2,3]Department of Computer Science & Engineering
[1,2,3]K.S.R College of Engineering, Tamilnadu, India

Abstract— This paper provides an extensive overview of existing research in the field of software refactoring. To make a computer program more readable and maintainable, refactoring technique is used. Code smells may be considered as a potential problem in the software design which needs to be refactored, in order to enhance the quality of a particular software product. Locating a code smell is hard and cumbersome task, because code smells are usually scattered across the source code. Smell detection tools and techniques have been proposed to help improve the user experience and quality of the software. Recent researches towards smell detection suggest many automated smell detection tools may help the humans to detect code smells. The results of the above experiments focuses on what the current smell detection can tools do and relevant areas of further improvement.

Key words: Software Design, Code Smell Detection

I. Introduction

Software design is the process of producing solution to analysed requirements and Specifying detail of how system will meet requirements. Software requirements analysis (SRA) is considered as the important part of the software design for accessing user requirements. Software design will help the programmer in software coding and Implementation by transforming user requirements to a suitable form. Software design can be considered as creating a solution to a problem in hand with available capabilities. The ultimate goal of research in software engineering (SE) is to improve the productivity and quality associated with both the processes and products of software development. To mitigate risks and to facilitate the achievement of the overall software system, the quality issues should be considered very early in the software development process. An overall commitment to quality, Creative skill, past experience and a sense of what makes “good” software, are critical success factors for a competent design.

Code smells are indicators which may pinpoint something wrong in the code. Code smells can occur both in the production code as wells as test code. Data class, Divergent Change, Shotgun Surgery, Parallel Inheritance, Blob, and Feature Envy are certain types of code smells taken into consideration. Refactoring keeps the developer ready for change and gives a comfort zone to do the changes. Refactoring is the single most important factor in achieving agility.

The main goal of this work is to propose a technique for Detecting code smell in open source software and measure the quality based on certain metrics. Quality is defined as a set of features which measures the characteristics of a product that bear on its ability to satisfy stated or implied needs. Different perspectives of quality can be considered: - user view, as the quality of the final product; - developer view, as the quality of the intermediate products generated by the different stakeholders during the development process; - end-user manager view, as the marketing requirements. The overall quality of a product can be the expressed by a combination of the different views.

Code Smell	Description
1. Data Class	Classes with fields which may get and set methods for fields and nothing else
2. Shotgun Surgery	A single Change in a class requires a lot of little changes in several classes.
3. Blob	One large class monopolizes the behavior of a system while the other classes primarily encapsulate data.
4. Parallel Inheritance	Making of a subclass in one class also have to make subclass of another class.
5. Feature envy	A method seems to be more interested in another class than its actual residing class.
6.Divergent change	One class is commonly changed in different ways for different reasons.

Table 1: Types of Code Smells

II. Related Work

Code Smell detection has a promising impact on software evolution. Several works have been proposed for code smells detection. The approach may range from manual approaches, techniques which are based on inspection, heuristics related to metric based. Code smells and–or antipatterns are identified based on the sets of rules defined on various metrics. Rules may be defined using fuzzy logic and then be executed by means of a rule-inference engine or using visualization techniques.

Seonah Lee et al. [2] proposed Mining programmer Interaction histories (MI) scheme to discover the context of edit files using view histories. In order to enhance developer productivity, Recommendation systems are usually intended for recommending files to edit. The system mines association rules in software revision histories. Edit location identification process is not supported. Regarding the mining of revision histories relating to the software, researchers have focused on association rule mining, which finds rules among the presence of items in earlier transactions.

Gabriele Bavota et al. [4] focuses on the system analyzes the impact of Application Program Interface (API) changes and faults in Android environment. The goal of this study is survey Android developers, with the purpose of understanding to what extent they experience problems when using APIs and how much they consider these problems to be related with negative user ratings comments. Hence, the study quality mainly focuses on perception of the developer ability for the impact change- and fault-prone APIs found on the apps’ user ratings. Design factors are not focused in the system.

Abdelilah Sakti et al. [5] address a new automated SBST (search-based software test-data generation) approach for unit-class testing to achieve high code coverage. The proposed instance generator allows our approach to better explore the search space, reaching more test targets, thus increasing code coverage in less time. To improve over random testing, global and local search algorithms have been implemented in several ways. Despite of all the features, code smell discovery was not performed.

David Notkin et al. [3] presented a technique called codebase replication which simplifies the implementation of continuous analysis tools by converting an existing offline analysis into an IDE-integrated. Isolation and currency are considered as the two desirable properties of a continuous tool. Codebase Replication creates and maintains a copy of the developer's codebase. The analysis runs on the copy codebase without being disturbed by the changes made by the developer. Version based history level analysis was not performed.

Wael Kessentini et al. [7] consider detection of code-smells as a distributed optimization problem. During the process of optimization dissimilar methods are combined in parallel to find a consent regarding the detection of code-smells. In order to perform the above stated basic idea Parallel Evolutionary algorithms (P-EA) is used. In P-EA many evolutionary algorithms with different adaptations such as fitness functions, solution representations, and change operators are executed, in a parallel cooperative manner, to serve a common goal called code-smell detection.

Gabriele Bavota et al. [8] proposed a technique to identify opportunities for Move Method refactoring and remove the Feature Envy bad smell from source code. RTM (Relational topic model) is used by an approach called Methodbook to analyze both structural and textual information gleaned from software to better support move method refactoring. The internal structure of the software system undergoes continuous changes during software maintenance and evolution. Due to these modifications the source code gets drifted away from its original design, thus deteriorating its quality, including cohesion and coupling of classes. Several refactoring methods have been proposed to overcome this problem.

Chu-Ti Lin et al. [9] explored a rate based queuing simulation model by integrating testing and debugging process in software lifecycle. The main motive is to characterize the activities required for debugging the open source software projects. The Operations should be carried out in a consistent manner due to the increasing reliability placed on the software systems. OSS approach has good potentials to reduce the development cost, widen the usage, and enhance the reliability of the software product. The importance of code smell detection is not considered.

Fabio Palomba et al. [1] proposed a Historical Information for Smell detection(HIST), an approach exploiting change history information to detect instances of five different code smells, namely Divergent Change, Shotgun Surgery, Parallel Inheritance, Blob, and Feature Envy. Code smells are detected using structural information identified from version histories. Structural and lexical information are analyzed with rule definitions to discover the code smells. Rules are represented with metric factors with threshold values. Detection algorithm uses the code components such s methods and classes for code smell detection process. Apriori algorithm is used to extract code smells from change history data values. Minimum support and confidence values are used to filter out the code smells. Detecting and removing the code smells help developers to easily understand and enhance source code.

III. Detection Scheme for Code Smells

JDeodorant is an Eclipse plug-in which so far identifies four main types of code smells using a set of quality metrics: God class, feature envy(FE),Type checking and Long method. JDeodorant consists of many other features such as transforming expert knowledge to fully automated processes, prior evaluation of effect for each suggested solution, design problems can get solved by user guidance and user friendliness. Batch processing mode is considered as a new feature in JDeodorant. In this mode, all open Java projects can be analyzed without user interaction and no interaction with eclipse interface is required.

A. inCode:

inCode is also an eclipse based plug-in for detection of code smells. If programmer writes any bad structure, during programming than it shows these smells in the shape of red color blocks along with code as, "eclipse show error" and warnings. inCode detects 4 bad smells *i.e.* Feature envy, God class, Duplicate code and Data class.

B. inFusion1:

This is a commercial standalone tool for Java. Detects God Class, God Method, and Feature Envy.

C. PMD3:

This is an open-source tool for Java and an Eclipse plugin that detects many problems in Java code, including two of the smells of our interest: God Class and God Method. The detection techniques are based on metrics.

D. JDEvAn:

(Java Design Evaluation and Analysis) is an Eclipse plug-in. It evaluates a design evaluation history of software system and provides the information about the system history. This automated framework is used to refactoring the code smells occurs in the java source programs. This tool support Java version 6 source codes as input and produces refactored source code

IV. Conclusion

In this paper the general focus relay upon the detection of code smells. During the detection step itself the importance of code smells gets analyzed using previous code changes. Detection of code smells usually symbolize to find a solution for correcting the detected code smells. The code smells are prioritized based on the metrics and inter-smell relations among the code smells. Detecting and prioritizing the corresponding code smell will make the software efficient and maintained at high rate.

References

[1] Fabio Palomba, Gabriele Bavota, Massimiliano Di Penta, Rocco Oliveto, Denys Poshyvanyk and Andrea De Lucia, "Mining Version Histories for Detecting Code Smells", IEEE Transactions On Software Engineering, Vol. 41, No. 5, May 2015, pp.462-489.

[2] Seonah Lee, Sungwon Kang, Sunghun Kim and Matt Staats,"The Impact of View Histories on Edit

Recommendations", IEEE Transactions On Software Engineering, Vol. 41, No.3, March 2015, pp.314-330.

[3] Kivancs Mu slu,Yuriy Brun,Michael D. Ernst and David Notkin, "Reducing Feedback Delay of Software Development Tools via Continuous Analysis", IEEE Transactions On Software Engineering, Vol. 41, No. 8, August 2015, pp.745-763.

[4] Gabriele Bavota , Mario Linares-V_asquez, Carlos Eduardo Bernal-C_ardenas, Massimiliano Di Penta, Rocco Oliveto and Denys Poshyvanyk," The Impact of API Change and Fault-Proneness on the User Ratings of Android Apps", IEEE Transactions On Software Engineering, Vol. 41, No. 4, April 2015, pp.384-407.

[5] Abdelilah Sakti, Gilles Pesant and Yann-Gael Gu_eh_eneuc, "Instance Generator and Problem Representation to Improve Object Oriented Code Coverage", IEEE Transactions On Software Engineering, Vol. 41, No. 3, March 2015, pp.294-313.

[6] Theodore Chaikalis and Alexander Chatzigeorgiou,"Forecasting Java Software Evolution Trends Employing Network Models", IEEE Transactions On Software Engineering, Vol. 41, No. 6, June 2015, pp.582-602.

[7] Wael Kessentini, Marouane Kessentini, Houari Sahraoui, Slim Bechikh and Ali Ouni, "A Cooperative Parallel Search-Based Software Engineering Approach for Code-Smells Detection", IEEE Transactions On Software Engineering, Vol. 40, No. 9, September 2014, pp.841-861.

[8] Gabriele Bavota, Rocco Oliveto, Malcom Gethers, Denys Poshyvanyk and Andrea De Lucia," Methodbook: Recommending Move Method Refactorings via Relational Topic Models", IEEE Transactions On Software Engineering, Vol. 40, No. 7, July 2014, pp.671-694.

[9] Chu-Ti Lin and Yan-Fu Li,"Rate-Based Queueing Simulation Model of Open Source Software Debugging Activities", IEEE Transactions On Software Engineering, Vol. 40, No. 11, November 2014, pp.1075-1099.

[10] Miryung Kim, Thomas Zimmermann and Nachiappan Nagappan, "An Empirical Study of Refactoring Challenges and Benefits at Microsoft", IEEE Transactions On Software Engineering, Vol. 40, No. 7, July 2014, pp.633-649.

