Proceedings of the International Conference on Intelligent Computing and Control Systems (ICICCS 2019)
IEEE Xplore Part Number: CFP19K34-ART; ISBN: 978-1-5386-8113-8

IDENTIFICATION OF CODE SMELL USING MACHINE LEARNING

Jesudoss A.
Assistant Professor:
Department of Computer Science and Engineering
Sathyabama Institute of Science and Technology
Chennai, Tamil Nadu, India
jesudossas@gmail.com

S.Maneesha, T.Lakshmi naga durga
Students:
Department of Computer Science and Engineering
Sathyabama Institute of Science and Technology
Chennai, Tamilnadu, India
maneesha.sudireddy@gmail.com
aswinitatikayala@gmail.com

***ABSTRACT*: Code smells are used to improve the quality of the software. Code smell detection detects the code problems like long method, large class, lazy class, long parameter list, feature envy, primitive obsession detector and too many literal detectors present in the code. In this work, two algorithms are used namely Support Vector Machine (SVM) and Random Forest algorithm. Support Vector Machine acts as classifier and the Random Forest algorithm are used for predicting the range of data. Decision making technique is used to identify the various problems present in the code. Code smell detection is a testing tool and it is mainly used by the developers, when the size of the code becomes unmanageable for manual detection. It also identifies deeper problems like syntax error, runtime error. It gives the output by analyzing the code in six different modules in this work.**

Key Words: Code Smell, Bloated Code Detector, Lazy Class Detector, Feature Envy, Large Class, Long method, Primitive Obsession Detector, Bad Smells.

I. INTRODUCTION

Now-a-days there are so many tools available for detecting code smells in the programming like Eclipse plug-in, which is a stand-alone application. The main issue is that many are facing problems in detecting the code smell and this detection is also very difficult. This concept is mainly introduced by Fowler, who defined different kinds of smells. From thereafter, many authors have discovered many different types of code smells. They discovered new code smells for improving the quality of software. The code smells of bloated code detector, lazy class detector, primitive obsession detector plays an important role in finding the bad smells.

A. Code Smell

Fowler identified 22 code smells for improving Structure of code. In these 22 code smells, some of the code smells include the real problems in the code such as long parameter list, etc.

B. Classifications of Smells

The code smells proposed by Fowler aim to identify the bad smells present in the code. There were 22 code smells identified that are long method, large class, primitive obsession, long parameter list, data clumps, shotgun surgery, duplicate code, lazy class, data class, dead code, feature envy, inappropriate intimacy, message chains, too many literal detectors, switch statements, refused request, temporary field, change preventers, alternative classes with different interfaces, parallel inheritance hierarchies, speculative generality and dispensable.

II. LITERATURE SURVEY

There are two metrics. They are Cohesion Coupling and size metrics. Based on these metrics, the bad smells identified into two sections [1]. The main advantage of code smell is to develop the code in an efficient way [2]. There is a difference in code smell that is instance of smell in data sets [3]. Software Product Line technique proposed by Ramon Abilio et al provides more accuracy and hence it is used in many detection techniques [4]. Neetu Faujdar et al proposed a bad smell technique in highly configurable software. Hence, the quality of software is very high for bad smells. It was developed in Java [5]. Sangeetha et al proposed a code smell detection that was implemented only for Lazy classes using J Smell which is written in Java [6]. Mohamed Wiem Mkaouer et al identified through research about various questions from many websites. Are developers properly developed? [7]. Alessandro Garcia et al implemented large class and implemented using J Deodorant and J Spirit [8]. Rafael de mello et al improved different abstraction levels like gate level, register transfer level. But didn't improved

code smells [9]. Jose Pereira dos Reis proposed find smells that user can compose. It cannot run without input source code [10]. Bruno L. Sousa et al investigated on many papers and many issues like test smells as design issues [11]. Massimiliano Di Penta et al work includes crowdsourcing this can improve source code [12]. Roberto Oliveira et al implemented long method using Binary logistic regression model [13]. Zhifei Chen et al used many techniques like extraction method, polymorphism, Context-refactoring to improve the quality of code smell [14]. Software product lines (SPL) technique is used. God class, Shot gun surgery are implemented [15]. Massimiliano Di Penta et al used K-means clustering algorithm. This compares with other smells and refracts on source code [16]. Shizhe Fu et al aim is to detect three code bad smells by using mining software evolutionary data they are duplicated code, shotgun surgery, and divergent change [17]. Francesca Arcelli Fontana implemented code debt that can be more smell intensity [18]. Glauco de Figueiredo Carneiro implemented two methods they are Crowd smelling and smelly maps by using neural network algorithm [19]. If the code contains a potential design problem like severity problem that was identified by Fabio Palomba et al [20]. There are many roots in detecting code smells like lazy class, long method etc. For this purpose, Sign Square Contour Algorithm (SSCA) algorithm is used [21]. Based on machine learning the work has done, many detection tools are developed such as lazy class long method [22]. Francesca Arcelli Fontana et al implemented only two smells namely lazy class and god class [23]. Eva van Emden et al aim is to examine software quality and the software inspection technique is used [24]. Francesca Arcelli Fontana et al aim is to compare tools and advantage in this the work [25]. Marco Zanoni et al used standard object-oriented technique to improve the correlations in code smell [26]. Jesudoss et al proposed a dynamic authentication technique for web applications [27]. NaouelM oha et al didn't compare and improves quality and performance [28]. Glauco de F. Carneiro et al used distribution map technique which is used to analyze and visualize [29]. Yovan Felix et al used image processing technique to identify the data and implemented in Java [30].

III. PROPOSED SYSTEM

The technique for detecting the code smells by using the developer's context and used the support vector machine algorithm. By using automated impact analysis for analysing and detecting the code smells. The output of detecting the code smell is in six different kinds of smells. They are Bloated Code Detector, Lazy Class Detector, Primitive Obsession Detector, Duplicated Code Detector, Feature Envy Detector and Too Many Literal Detectors. In the bloated code detector, it detects the code having the long method, large class and long parameter list. It detects the code and displays how many lines of long method with the file name, class name and method name. By applying the smell on that long method, it displays only that long method code.

A. TYPES OF CODE SMELLS

a) Duplicated Code: Duplicated Code is nothing but if the same code occurs more than once then that code is said to be as duplicated code. Detecting the duplication is very easy. Identifying the duplicates are done by measuring the duplicated code percentage in line. The problem occurs in the measurement of the code smell which are present in different types.

b) Feature Envy: The method which shows more interest in other class and the one which uses the methods of other classes are called as the feature Envy.

c) Large Class: The class which consists of too many variables and too many methods i.e., a class that deals with more variables or methods is named as large class.

d) Long Method: The method which is too long and is very difficult to understand the code is named as long method.

e) Long Parameter List: The code which contains too long parameters and is very difficult to understand is named as long parameter list.

f) Smell Detection: The smell detection defines how the smells can be detected and it discusses about the problems occur in the code like long parameter list, primitive obsession detector, etc.

g) Lazy Class: The class in which consists of the average method count is less than 3 and average variable count is less than 4, then that class is classified as the Lazy class i.e., that class will have too low classes or small classes.

h) Primitive Obsession Detector: If the number of parameter values exceeds the given number of corresponding variable value then it is classified as the primitive obsession detector.

B. Advantages

- The less time is taken for detecting the code smells.
- The main advantage is that it will detect the code automatically.
- Many code smells are implemented like long parameter list, duplicated code, and Primitive obsession detector.

IV. IMPLEMENTATION

Code Smell is implemented in Java using Net Beans IDE. The input for code smell is source code. First choose the code and then choose the directory then click the process, and then it shows the summary of file count, method count, constructor count, variable count and class count. Then go to the file details detect various classes, methods, variables, lazy classes, feature envy, duplicated code and bloated code. For this detecting process, in each code smell support vector machine is used as classifier and random forest algorithm for displaying the code. For each and every code smell, there are rules & syntaxes. Hence for generating rules, decision making technique is used.

Using code detection techniques, one can detect the bloated code detector, lazy class detector, feature envy detector, primitive obsession detector, duplicated code detector and too many literal detectors. In bloated code detector one can detect the long method, large class and long parameter list. Apply the smells on long method then automatically it smells the code and displays the code of how many lines of long method present in the code with the file name, method name and class name. Hence, the process of smelling the classes and methods in code takes place automatically using these techniques.

V. SYSTEM ARCHITECTURE

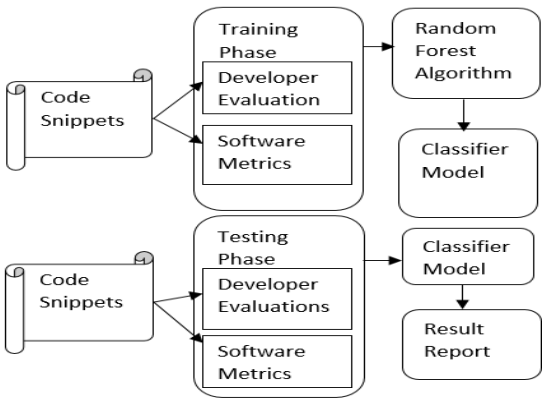

Fig 1: Training Phase and Testing Phase of Code smell.

According to this Code Smell Detection architecture, the code snippet consists of the programming syntax. In this work, Java programming is used and hence, the syntax of the java programming takes place in code snippets. There are two phases takes place in code smell detection, one is Training phase and another one is Testing phase (fig 1). After identifying the input source code, file details are verified and detects the code smells. In verifying the file details, the different classes, separate file content and variable definition are considered to be important. The six levels of code evaluations takes place in detecting code smells. In this process, all types of code smells are detected and given to developer for evaluation. After detecting all these smells, the support vector machine, random forest algorithms and the decision making techniques should be followed. The support vector machine (SVM) is used for identifying the classification and regression challenges, and the decision making techniques is also used for addressing the issue in the code. The whole result code of training phase goes to the classifier model. Finally, in testing phase, the final result occurs in the result report.

VI. COMPARISION

TABLE I.

S.No	Existing System	Proposed System
1	Less code smells are implemented like large class, feature envy, lazy class	More code smells are implemented like bloated code detector, lazy class, long method, primitive obsession detector, duplicated code and too many literal detectors
2	Code is difficult to understand for the developers	Easy to understand for developers
3	Developing more attributes	Classifying and developing more attributes
4	Time complexity for detecting the code is more	Time complexity for detecting the code is less

VII. RESULTS

A. Bloated Code Detector

When the particular entry in Long method is clicked, it shows the details of the smells of Long method, Large class and Long parameter list.

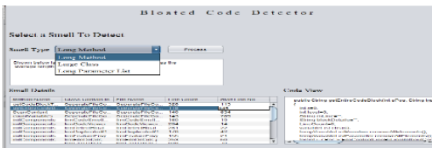

Fig. 2: Detecting code Smell in bloated code detector

When the line count is selected, then from the sample program the line count is shown as in Fig. 2.

B. Lazy Class Detector

Average method count is less than 1 and average Variable count is less than 5 then it comes under Lazy class detector.

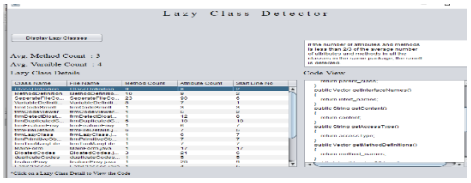

Fig. 3: Detecting Code Smell in Lazy Class Detector

If the user clicks on any one of the lazy class then it display the code from where lazy class gets Started and ended as shown in Fig. 3.

C. Primitive Obsession Detector

The Primitive Obsession Detector ensures the number of primitive variables and the numbers of parameter variables exceed the maximum number of the corresponding value.

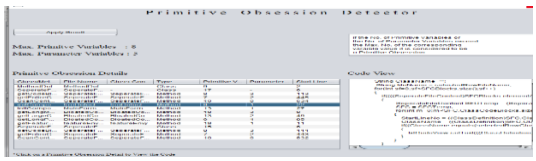

Fig. 4: detecting code smells in Primitive Obsession Detector.

If the user clicks on any one of the primitive Obsession Detector it displays the code of that particular Exceeded number present in the code as shown in Fig. 4.

D. Duplicated Code Detector

In duplicated code detector, when separate file such as content.java is opened and class definition.java is selected, then sample code will be displayed. The duplicated lines of code will be identified.

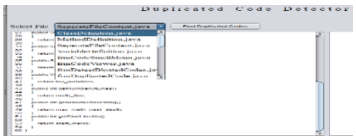

Fig. 5: detecting the code smell in Duplicated Code Detector

In this only the code which is repeated more than once that only Displayed in Duplicated code Detector as shown in Fig. 5.

E. Feature Envy Detector

By applying smells, feature envy details will be shown as method name, file name, class name and starting line number of the code with number of lines.

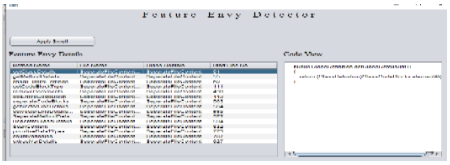

Fig. 6: Detecting code smell in feature envy detector

By selecting start line number, lines of code will be displayed which are having the methods used by another class in sample code as shown in Fig. 6.

F. Too Many Literals Detector

By applying smell, class or method, file name, type, primitive variable, parameter, start line number will be displayed.

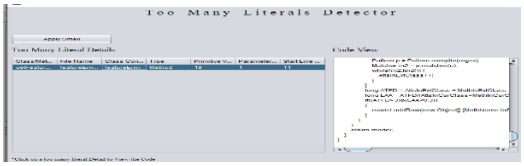

Fig. 7: Detecting the code smell in Too many literals detector

By selecting code view code lines of can be identified as shown in Fig. 7.

VIII. CONCLUSION

The Code Smell Detection has been implemented in an efficient way and it remains very useful for developers, to optimize the code quickly. Hence, it helps developers to produce a code with less complexity and minimal duplication.

REFERENCES

[1] Elvira-Maria Arvanitou, Structural Quality Metrics as indicators of the long Method Bad Smell: An Empirical Study (IEEE), 2018.

[2] Tushar Sharma, Detecting and Managing code smells: Research and Practice (IEEE), IEEE/ACM 40th international conference on software engineering: companion (ICSE-companion), 2018.

[3] Dario Di Nucci; Fabio palomba; Damian A. Tamburri; Alexander serebrenik; Alexander De Lucia Detecting Code Smells using Machine Learning Techniques: Are We There Yet? (IEEE), IEEE 25th International Conference on Software Analysis, Evolution and Reengineering (SANER), 2018.

[4] Ramon Abílio; Julianaa Padilha; Eduardo Figueiredo: Heitor Costa Detecting code smells in Software Product Lines -- An Exploratory Study, 12th International Conference on Information Technology new generation, 2018.

[5] Neetu Faujdar; Kshitij Srivastav; Megha Gupta; Shipra Saraswat, Detecting Strategies of Bad Smells in Highly Configurable Software,8th International Conference on Cloud Computing, Data Science & Engineering (Confluence), 2018.

[6] M.Sangeetha; P.Sengottuvelan Systematic Exhortation of Code Smell Detection Using J Smell for Java Source Code (IEEE), International Conference on Inventive Systems and Control (ICISC), 2017

[7] Mohamed Wiem Mkaouer, A Permission Smell Detector for Android Applications (IEEE), 2017.

[8] Alessandro Garcia, Towards Effective Teams for the Identification of Code Smells (IEEE), 2017.

[9] Rafael de Mello ; Roberto Oliveira ; Leonardo Sousa ; Alessandro Garcia Towards Effective Teams for the Identification of code smell IEEE/ACM 10th International Workshop on Cooperative and Human Aspects of Software Engineering (CHASE), 2017.

[10] José Pereira dos Reis; Fernando Brito e Abreu; Glauco de F. Carneiro Code Smells detection 2.0: Crowds melling and visualization, 12th Iberian Conference on Information Systems and Technologies (CISTI), 2017.

[11] Bruno L. Sousa ; Priscila P. Souza ; Eduardo M. Fernandes ; Kecia A.M. Ferreira; Mariza A.S. Bigonha, Find Smells: Flexible Composition of Bad Smells Detection Strategies ,IEEE/ACM 25th International Conference on Program Comprehension (ICPC), 2017.

[12] Massimiliano Di Penta, An Empirical Investigation into the Nature of Test Smells (IEEE), 2016.

[13] Roberto Oliveira, When More Heads Are Better than One? Understanding and Improving Collaborative Identification of Code Smells (IEEE), 2016.

[14] Zhifei Chen; Lin Chen; Wanwangying Ma; Baowen Xu Detecting Code Smells in Python Programs International Conference on Software Analysis, Testing and Evolution (SATE), 2016.

[15] Diego Cedrim, Context-Sensitive Identification of Refactoring Opportunities, IEEE, 2016.

[16] Massimiliano Di Penta, When and Why Your Code Starts to Smell Bad, IEEE, 2015.

[17] Shizhe Fu; Beijun Shen Code Bad Smell Detection through Evolutionary Data Mining, ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM), 2015.

[18] Francesca Arcelli Fontana ; Vincenzo Ferme ; Marco Zanoni ; Riccardo Roveda Towards a prioritization of code debt: A Code Smells Intensity, IndexIEEE 7th International Workshop on Managing Technical Debt (MTD), 2015.

[19] Glauco de Figueiredo Carneiro, Streamlining Code Smells: Using Collective Intelligence and Visualization (IEEE), 2014.

[20] Fabio Palomba ; Gabriele Bavota ; Massimiliano Di Penta ; Rocco Oliveto ;Andrea De Lucia Do They Really Smells Bad? A Study on Developers' Perception of Bad Code Smells, IEEE International Conference on Software Maintenance and Evolution, 2014

[21] Fabio Palomba ; Gabriele Bavota ; Massimiliano Di Penta; Rocco Oliveto; Andrea De Lucia; Denys Poshyvanyk, Detecting bad smells in source code using change history information, 28th IEEE/ACM International Conference on Automated Software Engineering (ASE), 2013.

[22] Francesca Arcelli Fontana; Marco Zanoni; Alessandro Marino; Mika V. Mäntyla Code Smells Detection: Towards a Machine Learning-Based Approach IEEE International Conference on Software Maintenance, 2013

[23] Amin Milani Fard; Ali Mesbah JSNOSE: Detecting java script Code Smells IEEE 13th International Working Conference on Source Code Analysis and Manipulation (SCAM), 2013.

[24] Eva van Emden; Leon Moonen, Assuring Software Quality by Code Smell Detection(IEEE),19th Working Conference on Reverse Engineering, 2012.

[25] Francesca Arcelli Fontana, An experience report on using code smells detection tools (IEEE), 2011.

[26] Marco Zanoni, On investigating code smells correlations, IEEE, 2011.

[27] Isela Macia; Alessandro Garcia; Arndt von Staa, Defining and Applying Detection Strategies for Aspect-Oriented code smell, Brazilian Symposium on Software Engineering, 2010.

[28] NaouelM oha ; Yann-Gael Gueheneuc ; Laurence Duchien ; Anne-Francoise Le Meur DECOR: A Method for the Specification and Detection of Code and Design Smells (IEEE) Transactions on Software Engineering, 2010.

[29] Glauco de F. Carneiro; Marcos Silva; Leandra Ma ; Eduardo Figueiredo; Claudio Sant'Anna; Alessandro Garcia; Manoel Mendonca Identifying Code Smells with Multiple Concern Views, Brazilian Symposium on Software Engineering, 2010.

[30] Steffen M. Olbrich; Daniela S. Cruzes; Dag I.K. Sjøberg, Are all Code Smells harmful? A study of God Classes and Brain Classes in the evolution of three open source systems IEEE International Conference on Software Maintenance, 2010.